DTIC FILE COPY

Copy 12 of 29 copies

(2)

AD-A200 452

IDA MEMORANDUM REPORT M-461

AN Ada/SQL IMPLEMENTATION KIT

Bill R. Brykczynski
Kerry Hilliard

DTIC
ELECTE
OCT 0 4 1988
S D

April 1988

Prepared for
WIS Joint Program Management Office

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

88 10 3 047

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

IDA Log No. HQ 88-033318

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Memorandum Reports

IDA Memorandum Reports are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Memorandum Reports is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This Memorandum Report is published in order to make available the material it contains for the use and convenience of interested parties. The material has not necessarily been completely evaluated and analyzed, nor subjected to IDA review.

Approved for public release/unlimited distribution; unclassified.

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Public release/distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Memorandum Report M-461	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Institute for Defense Analyses 6b OFFICE SYMBOL: IDA	7a NAME OF MONITORING ORGANIZATION OUSDA, DIMO
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	7b ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311
8a NAME OF FUNDING/SPONSORING ORGANIZATION: WIS JPMO 8b OFFICE SYMBOL (if applicable): WJPMO/DXP	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) Room 5B19, The Pentagon Washington, D.C. 20330-6600	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-W5-206 / WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) An Ada/SQL Implementation Kit (U)	
12 PERSONAL AUTHOR(S) Bill R. Brykczynski, Kerry Hilliard	
13a TYPE OF REPORT: Final 13b TIME COVERED: FROM _____ TO _____	14 DATE OF REPORT (Year, Month, Day): 1988 April 15 PAGE COUNT: 88
16 SUPPLEMENTARY NOTATION	

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Ada programming language; Structured Query Language (SQL); Ada/SQL; WIS; interfaces; software; implementation; application scanner; application generator; Oracle database management system; software tools.

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this IDA Memorandum Report is to describe additional documentation for the application scanner tool described in IDA Memorandum Report M-460, *An Ada/SQL Application Scanner*. In M-461, two types of information are presented: (1) the identification and description of the particular software of the application scanner which may be modified if rehosted to a different environment and (2) additional documentation describing lower-level modules used to implement a major tool of the Ada/SQL system. Section 1 contains introductory and background material. Section 2 is a description of the application scanner system dependencies, with discussions of the standards directory and files, file extensions, naming conventions of files, tables, and columns and the debug options. Section 3 contains the two types of documentation for the application scanner, with an overview of each file and then the actual file documentation.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☑ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include area code)	22c OFFICE SYMBOL
Bill R. Brykczynski	(703) 824-5515	IDA/CSED

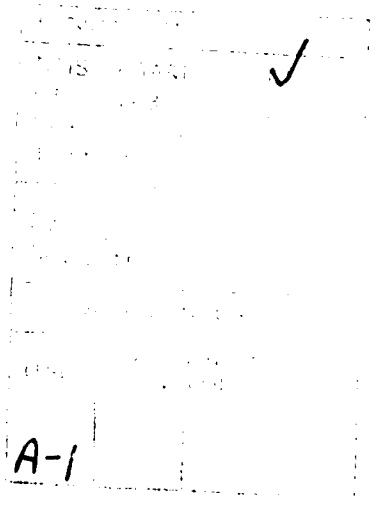
DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

IDA MEMORANDUM REPORT M-461

AN Ada/SQL IMPLEMENTATION KIT

Bill R. Brykczynski
Kerry Hilliard

April 1988



IDA
INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-W5-206

CONTENTS

PREFACE

The purpose of IDA Memorandum Report M-461, *An Ada/SQL Implementation Kit*, is to forward data developed in the course of investigation of the problems and requirements for rehosting an Ada/Structured Query Language (SQL) system.

The importance of this document is based on fulfilling the objective of Task Order T-W5-206, WIS Application Software Study, which is to support the idea that "programs using the Ada/SQL interface will be readily portable to other environments consisting of different hardware, operating systems, database management systems, etc." M-461 will be used to demonstrate this capability. As a Memorandum Report, M-461 is directed toward users who are concerned with how an Ada/SQL system is implemented and operates.

1. INTRODUCTION

The purpose of this IDA Memorandum Report M-461, *An Ada/SQL Implementation Kit*, is to describe additional documentation for the application scanner tool as described in [IDA 88c]. Two types of information are presented:

- The identification and description of the particular software of the application scanner which may be modified if rehosted to a different environment.
- Additional documentation describing lower-level modules used to implement a major tool of the Ada/SQL system.

1.1 SCOPE

Section 2 is a description of the application scanner system dependencies, with discussions of the standards directory and files, file extensions, naming conventions of files, tables, and columns, and the debug option.

Section 3 contains the two types of documentation for the application scanner, with an overview of each file and then actual file documentation. Section 6.1, "Overall Description," acts as table of contents for the documentation by listing the file names and a brief description of each one.

1.2 BACKGROUND

The documentation contained in this report should be used in conjunction with several other reports. A technical description of Ada/SQL, an interface between the Ada programming language and the database programming language SQL, is contained in [IDA 86] and [IDA 88a]. These memorandum reports describe the various components of an Ada/SQL application, and provide a formal specification for the Ada/SQL language. Second, [IDA 88b] describes an Ada/SQL implementation connected to the database management system (DBMS) Oracle®. Finally, [IDA 88c] provides guidelines for the use of a major tool contained in the Ada/SQL - Oracle implementation. The documentation of this tool, named the application scanner, is the subject of this report.

The application scanner is a tool which aids in the generation of subprograms necessary for an Ada/SQL application. It reads the various segments of an Ada/SQL application, determines which operators and routines are necessary for compilation, and creates a package which the user then 'with's into his application. The tool should be thought of as an application generator, not a preprocessor. An application generator typically creates a separate piece of software from some form of specification (e.g. requirements specifications, design specification, code, etc.), while a preprocessor transforms one piece of software into another.

The Oracle - Ada/SQL implementation consists of a fairly large amount of Ada software. As with most software produced under the direction of the WIS Joint Program Management Office, it was planned to be released into the public domain. With such a large piece of public domain software, it is anticipated that this implementation will be rehosted from the current Ada environment (DEC VAX™/VMS, DEC Ada compiler, Oracle DBMS), to different environments. As such, it was planned to create an implementation kit, which provides two types of information:

ORACLE is a registered trademark of Oracle Corporation.

- A rehost guide would provide all known implementation dependencies.

  These dependencies consist of code which may have to be changed in case of a rehost to a different environment. For example, the application scanner must be able to open and read various files associated with an Ada/SQL application. One compiler may require filenames to end with a '.a' file extension, while another may require filenames to end with '.ADA'.

- A more refined level of documentation for the application scanner.

While the scanner is documented well enough for the designers of the tool, the fact that the scanner would be placed in the public domain demanded a greater amount of source level commenting. As such, a review was made of all packages for the Oracle - Ada/SQL implementation. Those packages deemed lacking proper source level documentation for public domain software were then commented.

1.3 REFERENCES

The following references are cited in this document and used to supplement information required to understand the process of rehosting a major Ada software tool and a more detailed understanding of how the tool functions.

[IDA 86] Brykczynski, Bill and Fred Friedman. 1986. *Preliminary version: Ada/SQL: A standard, portable Ada-DBMS interface.* Alexandria, VA: Institute for Defense Analyses. IDA Paper P-1944.

[IDA 88a] Brykczynski, Bill, and Fred Friedman. 1988. *Ada/SQL binding specifications.* Alexandria, VA: Institute for Defense Analyses. IDA Memorandum Report M-362.

[IDA 88b] Brykczynski, Bill, Fred Friedman, and Kerry Hilliard. 1988. *An Oracle-Ada/SQL implementation.* Alexandria, VA: Institute for Defense Analyses. IDA Memorandum Report M-459.

[IDA 88c] Brykczynski, Bill, and Fred Friedman, Kevin Heatwole, and Kerry Hilliard. 1988. *An Ada/SQL application scanner.* Alexandria, VA: Institute for Defense Analyses. IDA Memorandum Report M-460.

VAX is a trademark of Digital Equipment Corporation.

2. APPLICATION SCANNER SYSTEM DEPENDENCIES

2.1 Standards Directory and Files

A directory must be created which will be used by the application scanner at execution time. The directory should contain the three standard Ada modules. The directory and files are defined in the compilation unit DDL_IO_DEFS_SPEC.ADA as constants and by compilation unit DDL_IO_DEFS.ADA as functions. The three files needed by the application scanner are STANDARD.ADA, CURSOR_DEFINITION.ADA, and DATABASE.ADA. The names minus the extension (which may vary from system to system) are defined by the constants STANDARD_-NAME, STANDARD_NAME_ADA_SQL, CURSOR_NAME, CURSOR_NAME_ADA_SQL, DATABASE_NAME and DATABASE_NAME_ADA_SQL. Should the names of the files change these constants would also have to change.

The complete path name of the file which defines the directory in which they reside are defined by the functions STANDARD_NAME_FILE, CURSOR_NAME_FILE, and DATABASE_-NAME_FILE. These functions return a string which is the fully qualified file name, minus an extension. The string can be fully qualified file name, such as "USU:[BBRYKCZYN.ORACLE.-STANDARDS]STANDARD" or it can be an environment reference (if the system supports such a thing) such as "ADASQL$ENV:STANDARD", on the VAX. When the environment reference is made, the environment must be set before execution of the application scanner. Environment is set on the VAX as follows:

ASSIGN USU:[BBRYKCZYN.ORACLE.STANDARDS] ADASQL$ENV

2.2 File Extensions

All systems must have file extensions as a part of the file name. This extension will be used to read files which the application scanner finds references to in the 'with' clauses and also to reference the standard files discussed above. In the compilation unit DDL_IO_DEFS_SPEC.ADA, the following four constants and variables define that extension to the system:

```
DOT_ADA_LEN             : constant POSITIVE := 4;
DOT_ADA_UPPER           : constant STRING := ".ADA";
DOT_ADA_LOWER           : constant STRING := ".ada";
DOT_ADA_DEFAULT         : STRING (1..DOT_ADA_LEN) := ".ADA";
```

The following here shows the current values used which are for the VAX system:

- DOT_ADA_LEN is the total length of the extension including a delimiter such as the period used here.
- DOT_ADA_UPPER is the extension in upper case, a constant so it cannot be changed during execution of the application scanner.
- DOT_ADA_LOWER is the extension in lower case, a constant so it cannot be changed during execution of the application scanner.
- DOT_ADA_DEFAULT is the case that is being used for the current file that the application scanner is reading. This will be set and may change during execution, depending on the rules used to process upper case only files, lower case only files, or a mixture of upper and lower case.

2.3 Case of File Names

File names can be in upper case, lower case, or mixed case. If all files are to be in upper case (as it is on the VAX since they are treated the same), the HOW_TO_DO_FILES is set to UPPER_CASE. If all files are to be treated as lower case, the flag is set to LOWER_CASE. In the case of the files where some files may be lower and some upper, the flag is set to AS_IS.

In the case of UPPER_CASE or LOWER_CASE file names taken from 'with' clauses, the filenames are converted to the corrected case before being accessed. The extension will also be upper or lower case. For mixed case, the file will be referenced using the case as it appears in the 'with' clause, and the extension will be the case of the first character of the file name. These flags are located in the compilation unit DDL_IO_DEFS_SPEC.ADA and can be set in the compilation unit MAIN.ADA, so that to change it only MAIN.ADA need be recompiled.

2.4 Case of Table and Column Names

Table names and column names must
reference the components of the underlying database. Some databases require all table and/or column names to be upper case, some require lower case, some recognize a difference if only the case is changed, and some treat only a case change as the same name. In the application scanner all table names will have to be defined as being upper case or lower case. All column names must be upper case or lower case, but this need not be the same as the table case. Oracle does not recognize case so the setting of the flags is not important. The flags CASE_OF_TABLES and CASE_OF_COLUMNS are located in the compilation unit DDL_IO_DEFS_SPEC.ADA and can be set in the compilation unit MAIN.ADA, so that to change it only MAIN.ADA need be recompiled.

2.5 Debug Option

There is also a debug option in the application scanner. If this is set in the compilation unit MAIN.ADA, debug comments will print out during execution. There is currently commented out code to ask if debug comments are desired. The flag can either be set on or off or the user asked if debugging is desired. It is set off in the operating version.

3. Application Scanner Documentation

This section of the report provides two types of documentation for the application scanner. First, a brief purpose is stated for each of the compilation units associated with the tool. Second, documentation developed for each module is listed. Within the actual application scanner code, the comments are spatially divided to provide documentation relating to specific portions of the code. However, for the purposes of this report, all comments are collected together regardless of location within the application scanner.

3.1 Overall Description

```
chartob.ada      -- post process data strucs for CONVERT_CHARACTER_TO_COMPONENT
chartos.ada      -- post process data strucs for CONVERT_CHARACTER_TO_COMPONENT
columnb.ada      -- COLUMN_LIST data structures and for making a chain of
                    database columns
columns.ada      -- COLUMN_LIST data structures and for making a chain of
                    database columns
comptob.ada      -- post process data strucs for CONVERT_COMPONENT_TO_CHARACTER
comptos.ada      -- post process data strucs for CONVERT_COMPONENT_TO_CHARACTER
convb.ada        -- post process data structure for CONVERT_TO functions
convs.ada        -- post process data structure for CONVERT_TO functions
corrb.ada        -- post process/info for correlation names
corrs.ada        -- internal & post process data structures for correlation
                    names
database.ada     -- DATABASE definitions for the Application Scanner
dbtypeb.ada      -- post process data strucs for strongly typed database types
dbtypes.ada      -- post process data strucs for strongly typed database types
ddladesb.ada     -- ADD_DESCRIPTOR_ROUTINES add various descriptors to various
                    chains
ddladess.ada     -- ADD_DESCRIPTOR_ROUTINES add various descriptors to various
                    chains
ddlarayb.ada     -- ARRAY_ROUTINES process the array section of a type
                    declaration
ddlarays.ada     -- ARRAY_ROUTINES process the array section of a type
                    declaration
ddlauthb.ada     -- SCHEMA_AUTHORIZATION_ROUTINES process the authorization
                    clause
ddlauths.ada     -- SCHEMA_AUTHORIZATION_ROUTINES process the authorization
                    clause
ddlcallb.ada     -- CALL_TO_DDL_ROUTINES routines to initiate the ddl reader
                    to process the DDL for an application scanner DML module
ddlcalls.ada     -- CALL_TO_DDL_ROUTINES routines to initiate the ddl reader
                    to process the DDL for an application scanner DML module
ddldefs.ada      -- DDL_DEFINITIONS defines the data structures used by the
                    ddl reader to keep track of the schema units and the
                    information which they contain
ddlderb.ada      -- DERIVED_ROUTINES process the derived section of a type
                    declaration
ddlders.ada      -- DERIVED_ROUTINES process the derived section of a type
                    declaration
```

```
ddldrivb.ada     -- DRIVER is the driver for the ddl reader section of the
                    application scanner - body
ddldrivs.ada     -- DRIVER is the driver for the ddl reader section of the
                    application scanner - specification
ddlendb.ada      -- END_ROUTINES process an end of package statement
ddlends.ada      -- END_ROUTINES process an end of package statement
ddlenumb.ada     -- ENUMERATION_ROUTINES process the enumeration section of a
                    type declaration
ddlenums.ada     -- ENUMERATION_ROUTINES process the enumeration section of a
                    type declaration
ddlerrb.ada      -- ERROR_ROUTINES handel an unknown error
ddlerrs.ada      -- ERROR_ROUTINES handel an unknown error
ddlextrs.ada     -- EXTRA_DEFINITIONS defines some data structures and
                    variables used by the ddl reader to keep track of things
                    during the processing of the schema units
ddlfltb.ada      -- FLOAT_ROUTINES process the floating point section of a
                    type declaration
ddlflts.ada      -- FLOAT_ROUTINES process the floating point section of a
                    type declaration
ddlfuncb.ada     -- FUNCTION_ROUTINES process the "function x is new
                    authorization identifier;" statement
ddlfuncs.ada     -- FUNCTION_ROUTINES process the "function x is new
                    authorization identifier;" statement
ddlintb.ada      -- INTEGER_ROUTINES process the integer section of a type
                    declaration
ddlints.ada      -- INTEGER_ROUTINES process the integer section of a type
                    declaration
ddliodefb.ada    -- IO_DEFINITIONS contains the functions which return the
                    names of the standard files.
ddliodefs.ada    -- IO_DEFINITIONS contains IO related data structures, type
                    declarations and variables and the functions which return
                    the names of the standard files.
ddlioerb.ada     -- IO_ERRORS these are the error routines used by SCHEMA_IO
                    for the io routines
ddlioers.ada     -- IO_ERRORS these are the error routines used by SCHEMA_IO
                    for the io routines
ddlioinb.ada     -- IO_INTERNAL_STUFF these are the routines used by SCHEMA_IO
                    to do the nitty grittys for the io routines
ddlioins.ada     -- IO_INTERNAL_STUFF these are the routines used by SCHEMA_IO
                    to do the nitty grittys for the io routines
ddlkeyb.ada      -- KEYWORD_ROUTINES identifies the SQL and ADA key words
                    which cannot be used as identifiers
ddlkeys.ada      -- KEYWORD_ROUTINES identifies the SQL and ADA key words
                    which cannot be used as identifiers
ddllistb.ada     -- LIST_ROUTINES form the chains which hold the identifiers
                    for type variable and record component (database columns)
                    declarations, for which type descriptors will be created
                    the remainder of the declatation statement is valid
ddllists.ada     -- LIST_ROUTINES form the chains which hold the identifiers
```

for type variable and record component (database columns) declarations, for which type descriptors will be created the remainder of the declatation statement is valid

ddlmain.ada -- MAIN for testing purposes this will drive the ddl reader (without adding all the other application scanner code in) with input from the terminal and will display all data structures created

ddlmainc.ada -- MAIN_CALL for testing purposes this will drive the ddl reader (without adding all the other application scanner code in) in the same manner that it will be called when the application scanner is executing

ddlnameb.ada -- NAME_ROUTINES validate identifiers

ddlnames.ada -- NAME_ROUTINES validate identifiers

ddlndesb.ada -- GET_NEW_DESCRIPTOR_ROUTINES create and initialize various elements of the data structures in which the ddl reader will store data

ddlndess.ada -- GET_NEW_DESCRIPTOR_ROUTINES create and initialize various elements of the data structures in which the ddl reader will store data

ddlpackb.ada -- PACKAGE_ROUTINES process a package declaration

ddlpacks.ada -- PACKAGE_ROUTINES process a package declaration

ddlrecb.ada -- RECORD_ROUTINES process a record declaration

ddlrecs.ada -- RECORD_ROUTINES process a record declaration

ddlsdesb.ada -- SEARCH_DESCRIPTOR_ROUTINES page thru the data structures and return pointers to or information about various descriptors

ddlsdess.ada -- SEARCH_DESCRIPTOR_ROUTINES page thru the data structures and return pointers to or information about various descriptors

ddlshowb.ada -- SHOW_ROUTINES print the information collected in the data structures by the ddl reader

ddlshows.ada -- SHOW_ROUTINES print the information collected in the data structures by the ddl reader

ddlsiob.ada -- SCHEMA_IO the io routines related to the schema units to open and close files, to read data from files and the terminal, to output data to files and the terminal, and to perform data conversions

ddlsios.ada -- SCHEMA_IO the io routines related to the schema units to open and close files, to read data from files and the terminal, to output data to files and the terminal, and to perform data conversions

ddlsub1b.ada -- SUBROUTINES_1_ROUTINES contain some of the subroutines used by the ddl reader

ddlsub1s.ada -- SUBROUTINES_1_ROUTINES contain some of the subroutines used by the ddl reader

ddlsub2b.ada -- SUBROUTINES_2_ROUTINES contain some of the subroutines used by the ddl reader

ddlsub2s.ada -- SUBROUTINES_2_ROUTINES contain some of the subroutines used by the ddl reader

```
ddlsub3b.ada      -- SUBROUTINES_3_ROUTINES contain some of the subroutines
                     used by the ddl reader
ddlsub3s.ada      -- SUBROUTINES_3_ROUTINES contain some of the subroutines
                     used by the ddl reader
ddlsub4b.ada      -- SUBROUTINES_4_ROUTINES contain some of the subroutines
                     used by the ddl reader
ddlsub4s.ada      -- SUBROUTINES_4_ROUTINES contain some of the subroutines
                     used by the ddl reader
ddlsubb.ada       -- SUBTYPE_ROUTINES process a subtype declaration
ddlsubs.ada       -- SUBTYPE_ROUTINES process a subtype declaration
ddltypeb.ada      -- TYPE_ROUTINES process a type declaration
ddltypes.ada      -- TYPE_ROUTINES process a type declaration
ddluseb.ada       -- USE_ROUTINES process a use statement
ddluses.ada       -- USE_ROUTINES process a use statement
ddlvarb.ada       -- VARIABLE_ROUTINES process a variable declaration
ddlvars.ada       -- VARIABLE_ROUTINES process a variable declaration
ddlvrbls.ada      -- DDL_VARIABLES variables used during the processing of
                     schema units
ddlwithb.ada      -- WITH_ROUTINES process a token in a with context clauses
ddlwiths.ada      -- WITH_ROUTINES process a token in a with context clauses
dummys.ada        -- dummy data structure entries with null strings for lists
enumb.ada         -- manage internal data structures for enum type overloading
enums.ada         -- manage internal data strucs for enumeration type overloading
exprb.ada         -- routines to process expression-type constructs
exprs.ada         -- routines to process expression-type constructs
fromb.ada         -- internal data structures for from clauses
froms.ada         -- internal data structures for from clauses
funcdefs.ada      -- definitions of SQL operations
genfuncb.ada      -- post process/info for expression-type unary & binary ops
genfuncs.ada      -- post process/info for expression-type unary & binary ops
indexb.ada        -- post process data strucs for generated index subtypes needed
indexs.ada        -- post process data strucs for generated index subtypes needed
indicb.ada        -- post process data structures for INDICATOR functions
indics.ada        -- post process data structures for INDICATOR functions
intob.ada         -- post process data structures for INTO procedures
intos.ada         -- post process data structures for INTO procedures
lexb.ada          -- lexical analyzer handles token input and diagnostic
                     reporting
lexs.ada          -- lexical analyzer handles token input and diagnostic
                     reporting
main.ada          -- the driver routine for the application scanner
nameb.ada         -- parsing of various types of names
names.ada         -- parsing of various types of names
pdtypeb.ada       -- functions to identify predefined (STANDARD or DATABASE)
                     types
pdtypes.ada       -- functions to identify predefined (STANDARD or DATABASE)
                     types
pgmconvb.ada      -- post process data strucs for L_CONVERT & R_CONVERT functions
pgmconvs.ada      -- post process data strucs for L_CONVERT & R_CONVERT functions
```

```
postb.ada        -- produce generated package (specification and body).
posts.ada        -- produce generated package (specification and body).
predefb.ada      -- post process data structure for optional predefined text
predefs.ada      -- post process data structure for optional predefined text
qualb.ada        -- post process data structures for qualified column specs
quals.ada        -- post process data structures for qualified column specs
resultb.ada      -- internal data struc for keeping track of function result
                    type
results.ada      -- internal data struc for keeping track of function result
                    type
scanb.ada        -- driver for DML processing of Ada/SQL Application Scanner
scans.ada        -- driver for DML processing of Ada/SQL Application Scanner
searchb.ada      -- routine to process a search condition
searchs.ada      -- routine to process a search condition
selecb.ada       -- post process data structures for various flavors of SELEC
selecs.ada       -- post process data structures for various flavors of SELEC
selectb.ada      -- miscellaneous routines for processing select, declare,
                    insert_into and fetch statements
selects.ada      -- miscellaneous routines for processing select, declare,
                    insert_into and fetch statements
semanb.ada       -- miscellaneous routines for semantic processing
semans.ada       -- miscellaneous routines for semantic processing
stmtb.ada        -- process the open, delete, update, close and package
                    statements
stmts.ada        -- process the open, delete, update, close and package
                    statements
syntacb.ada      -- miscellaneous syntactic processing routines
syntacs.ada      -- miscellaneous syntactic processing routines
tableb.ada       -- miscellaneous routines for handling table names
tables.ada       -- miscellaneous routines for handling table names
tblexprb.ada     -- process clauses related to table expressions
tblexprs.ada     -- process clauses related to table expressions
tentb.ada        -- internal data structure for the tentative function list
tents.ada        -- internal data structure for the tentative function list
txtprt.ada       -- print utilities
unqualb.ada      -- post process/info for unqualified names (tables & columns)
unquals.ada      -- post process/info for unqualified names (tables & columns)
withb.ada        -- post process data structures for library units to be with'ed
withs.ada        -- post process data structures for library units to be with'ed
```

3.2 File Documentation

3.2.1 File DDLDRIVS.ADA

```
-- this is the driver for the ddl reader section of the application scanner

-- PROCESS_SCHEMA_UNIT - the ddl reader will process the schema unit who's
-- name is input to this routine.
```

```
-- PROCESS_FULL_SCHEMA_UNIT - processes or continues to process the schema who's
-- name is supplied as input to this routine.

-- SET_UP_CURRENT_SCHEMA_UNIT - set or create as the current schema unit the
-- schema unit who's name is provided as input to this routine

-- WHICH_PROCESS - given a token and the schema we're processing, return an
-- enumeration type for which process to do
```

3.2.2 File DDLDRIVB.ADA

```
-- this is the driver for the ddl reader section of the application scanner

-------------------------------------------------------------------
--
-- PROCESS_SCHEMA_UNIT - the ddl reader will process the schema unit who's
-- name is input to this routine.  The input to this routine is the name of a
-- schema unit, which must correspond to a file name.  We process the three
-- standard files, STANDARD.ADA DATABASE.ADA and CURSOR_DEFINITION.ADA first
-- if they haven't already been done.  We then process this schema unit thru
-- the ddl reader.

-------------------------------------------------------------------
--
-- PROCESS_FULL_SCHEMA_UNIT - processes or continues to process the schema who's
-- name is supplied as input to this routine.
--
-- set up the current schema unit, which might be a new one or one that has
-- already been done or one currently in process.
-- we loop doing the following until reaching the end of a file
--    then till exhausting the schema units yet to do list
--
-- read the next token, which must be something we recgonise.
-- when the end of the file is reached the DONE flag is set
-- if we are already in the middle of withing, flag set, then we call
--    PROCESS_WITH to do the next with in line or look for ; as a clue to the
--    end of withing
-- if the token is use, package, end, type, subtype, function, or
--    schema_authorization we have special routines to process the whole
--    statement
-- if the token is anything else tell the user it's an error

-------------------------------------------------------------------------
--
-- SET_UP_CURRENT_SCHEMA_UNIT - set or create as the current schema unit the
-- schema unit who's name is provided as input to this routine
--
```

```
-- set up the current schema, either an old one that wasn't finished or a
-- new one in which case we have to open the file.
-- search the list of already done schema_units, if this one hasn't
-- been done set up new pointers for it, add it to the chain and
-- set the name and open an input stream.
-- and if it's not STANDARD.ADA then show withing and using of it

------------------------------------------------------------------
--
-- WHICH_PROCESS - given a token and the schema we're processing, return an
-- enumeration type for which process to do
```

3.2.3 File DATABASE.ADA

```
-- DATABASE definitions for the Application Scanner
```

3.2.4 File DDLDEFS.ADA

```
-- DDL_DEFINITIONS defines the data structures used by the ddl reader to keep
-- track of the schema units and the information which they contain

-- STATUS_SCHEMA describes the current status of the schema unit
--        PROCESSING  this is the current schema being processed
--        WITHING     this schema unit is temporarily on hold while the schemas
--                    in it's with clause are processed
--        DONE        the processing of this schema is complete
--        NOTOPEN     this schema unit has not yet been opened
--        NOTFOUND    this schema unit was not found and could not be opened

-- KIND_TYPE describes the type of component in the descriptor
--     A_TYPE        a type declaration
--     A_SUBTYPE     a subtype declaration
--     A_DERIVED     a derived declaration
--     A_COMPONENT   a component (column) of a record (database table)
--                   declaration
--     A_VARIABLE    a variable declaration

-- TYPE_TYPE describes the data type of the descriptor
--      REC_ORD       a record (database table) type descriptor
--      ENUMERATION   an enumeration type descriptor
--      INT_EGER      an integer type descriptor
--      FL_OAT        a floating point type descriptor
--      STR_ING       a string (character array) type descriptor
```

```
-- YET_TO_DO_DESCRIPTOR describes a schema unit who's processing has not yet
-- been compleated

-- SCHEMA_UNIT_DESCRIPTOR describes one schema unit

-- WITHED_UNIT_DESCRIPTOR describes a schema unit that appeared in the with
-- clause of another schema unit

-- USED_UNIT_DESCRIPTOR describes a schema unit that appeared in the use
-- clause of another schema unit

-- DECLARED_PACKAGE_DESCRIPTOR describes a package that appeared in a schema
-- unit

-- IDENTIFIER_DESCRIPTOR describes an identifier such as a variable name of
-- type name etc. which appeared in a schema unit

-- FULL_NAME_DESCRIPTOR describes the fully qualified name of an identifier,
-- including its package name

-- TYPE_DESCRIPTOR describes a declaration of a record, enumeration, integer
-- floating point or string entity encountered in a schema unit

-- LITERAL_DESCRIPTOR describes an enumeration literal found in a schema unit

-- ENUM_LIT_DESCRIPTOR describes an enumeration literal

-- FULL_ENUM_LIT_DESCRIPTOR describes a fully qualified enumeration literal

-- ENUM_LIT_NAME_STRING is the data type used to store enumeration literals

-- AUTH_IDENT_NAME_STRING is the data type used to store authorization
-- identifiers

-- LIBRARY_UNIT_NAME_STRING is the data type used to store schema names, withed
-- and used schema etc.

-- PACKAGE_NAME_STRING is the data type used to store the names of packages
-- described in schema units

-- RECORD_NAME_STRING is the data type used to store the name of records which
-- when defined in a schema unit must ba a database table

-- TYPE_NAME_STRING is the data type used to store the identifiers for type,
-- subtype, variable etc declarations

-- ENUMERATION_NAME_STRING is the data type used to store the identifiers
-- for enumeration declarations
```

```
-- subtypes for each of the different type descriptors

-- YET_TO_DO_DESCRIPTORs will form a chain of SCHEMA_UNIT_DESCRIPTORs on
-- which processing is incomplete

-- the SCHEMA_UNIT_DESCRIPTORs will form a chain of schema units that have
-- been processed

-- WITHED_UNIT_DESCRIPTORS form a chain within the SCHEMA_UNIT_DESCRIPTORs
-- of all schema units withed by that schema unit

-- USED_UNIT_DESCRIPTORS form a chain within the SCHEMA_UNIT_DESCRIPTORs
-- of all schema units used by that schema unit

-- DECLARED_PACKAGE_DESCRIPTORS form a chain within the SCHEMA_UNIT_DESCRIPTORs
-- of all packages declared within that schema unit

-- IDENTIFIER_DESCRIPTORS form a chain of all identifiers declared in all
-- schema units

-- FULL_NAME_DESCRIPTORS form a chain of all fully qualified identifier names
-- declared in all schema units

-- TYPE_DESCRIPTORS form a chain of all declarations of types, subtypes,
-- derived types, record components (columns of tables) and variables

-- LITERAL_DESCRIPTORs for a chain of enumeration literals within a
-- TYPE_DESCRIPTOR

-- ENUM_LIT_DESCRIPTORs form a chain of all enumeration literals found in
-- all schemas

-- FULL_ENUM_LIT_DESCRIPTORs form a chain of the fully qualified neame of all
-- enumeration literals found in all schemas
```

3.2.5 File DDLEXTRS.ADA

```
-- EXTRA_DEFINITIONS defines some data structures and variables used by the
-- ddl reader to keep track of things during the processing of the schema units

-- PROCESS_TYPE is the type of ddl statement being processed
--     ITS_WITH                  - found a with statement
--     ITS_ALREADY_WITHING       - reading the schema units to be processed
--                                 as withed units
--     ITS_USE                   - found a use statement
--     ITS_PACKAGE               - found a package declaration
--     ITS_END                   - found an end package declaration
```

```
--      ITS_TYPE                    - found a type declaration
--      ITS_SUBTYPE                 - found a subtype declaration
--      ITS_FUNCTION                - found a "function x is new authorizarion
--                                    identifier" statement, the only function
--                                    declaration permitted in the ddl reader
--      ITS_SCHEMA_AUTHORIZATION    - found a schema authorization statement
--      ITS_EOL                     - reached the end of the file that we're
--                                    processing
--      ITS_UNKNOWN                 - hit an unknown keyword
--      ITS_FINISHED                - the schema unit has been compleately
--                                    processed

-- NAME_TO_PROCESS_LIST forms a chain of identifiers of type LIST_NAME_STRING
-- to be processed.

-- COMPONENT_TO_PROCESS_LIST forms a chain of record components (database
-- table columns) identifiers of type LIST_COMPONENT_STRING to be processed.

-- HOLDING_COMPONENT_DESCRIPTOR forms a chain of component (database columns)
-- descriptors processed.

-- variables used during processing
```

3.2.6 File DDLIODEFS.ADA

```
-- IO_DEFINITIONS contains IO related data structures, type declarations and
-- variables and the functions which return the names of the standard files.

-- INPUT_RECORD and INPUT_STREAM is the structure to keep track of the input
-- being read from schema unit files

-- HOW_TO_DO_FILES_TYPE defines possibilities for the case of file names etc

-- SCHEMA_FROM defines possibilities for the initiation of a schema unit.  It is
-- either initiated from a call (CALLS) from the application scanner or from
-- the schema unit file (FILES) such as a withed schema unit, or it is UNKNOWN

-- standard_name_file is as the file name should be accessed, without extention
-- standard_name is the package name
-- standard_name_ada_sql is the nexted package name

-- cursor_name_file is as the file name should be accessed, without extention
-- cursor_name is the package name
-- cursor_name_ada_sql is the nexted package name

-- database_name_file is as the file name should be accessed, without extention
-- database_name is the package name
```

```
-- database_name_ada_sql is the nexted package name

-- dot_ada is the extention to be used with the files

-- how_to_do_files  - if upper_case all file names are converted to upper case
--                    if lower_case all file names are converted to lower case
--                    if as_is they are to be used as entered by the user
```

3.2.7 File DDLIODEFB.ADA

```
-- IO_DEFINITIONS contains the functions which return the names of the
-- standard files.

-- standard_name_file is as the file name should be accessed, without extention

-- cursor_name_file is as the file name should be accessed, without extention

-- database_name_file is as the file name should be accessed, without extention
```

3.2.8 File DDLWITHS.ADA

```
-- WITH_ROUTINES process a token in a with context clauses

-- PROCESS_WITH process the next with token, the string "with", a comma,
-- a semicolon or a library unit name (schema unit)
```

3.2.9 File DDLWITHB.ADA

```
-- WITH_ROUTINES process a token in a with context clauses

-------------------------------------------------------------------
--
-- PROCESS_WITH process the next with token, the string "with", a comma,
-- a semicolon or a library unit name (schema unit)
--
-- if the temp string is WITH and the WITHING flag is set, tell the user
--    that with is an invalid library unit name and don't process it
-- if the temp string is WITH and the WITHING flag is not set, then set it
--    if a package name had already been declared in the current schema or if
--       types or tables or variables have been declared tell them that
--       context clauses must be first, but go ahead and process the with
--       statement
```

```
--    return
-- if the temp string is a comma, just return
-- if the temp string is a semi colon change the WITHING flag to PROCESSING
--    and return
-- otherwise we have a library_unit_name to process

-- process here if temp string = comma or semi colon or WITH

-- do a withed library unit here:
-- get the withed library unit's schema if it's been declared before
-- find out if this schema unit has withed this library unit before
-- if we're trying to with ourselves tell the user and ignore this with

-- if there is no schema for this with get a new schema, add it to the schema
--      chain, and set it's name
-- if it hasn't been withed before by the current schema unit then add it
--      to the chain of withed stuff
-- do not process the withed library unit name if it is schema_definition,
--      instead mark this one as done and continue with next
-- however if it is anything except schema-definition and this schema is an
--      authorization package tell the user that's not valid
-- if the status of the withing unit is already done then we don't have to do
--      anything else wth it

-- put the current schema unit on hold (yet to do list)
-- set the withed unit schema as the current schema unit
-- then open the new current schema unit and return and process it
```

3.2.10 File DDLUSES.ADA

```
-- USE_ROUTINES process a use statement

-- PROCESS_USE read thru the use statement processing package names an either
-- context clause uses or non context clause uses

-- PROCESS_USE_CONTEXT process a package from a use context clause, which
-- means it must have been withed by a prior with statement

-- PROCESS_USE_NON_CONTEXT process a package from a use non context clause,
-- which means it may be a qualified package name or it may be a subpackage
-- name from a package that has already been withed and used

-- VALID_USE - make sure the package being used is valid and has been withed
```

3.2.11 File DDLUSEB.ADA

```
-- USE_ROUTINES process a use statement

------------------------------------------------------------------
--
-- PROCESS_USE read thru the use statement processing package names an either
-- context clause uses or non context clause uses
--
-- when we enter this routine the temp string will be use
-- if no withs have been done it's an error to do a use, print error and
--    skip to end of use clause
-- if no packages have been declared we're processing a context clause use
-- if a package has been declared we're processing a non context clause use

-- we loop and read the next token, either a comma, a semicolon or package
--         to use
-- if comma - ignore it
-- if semi colon - the use statement is done and we return
-- otherwise we have a package_name to process
-- if this schema is an authorization package the only "use" permitted
--         is for schema_definition.  Anything else print an error.
-- call the appropriate routine to check it's validity and set up the
--    visibility pointers describing it, this depends on if it's a context
--    use or a non context use

------------------------------------------------------------------
--
-- PROCESS_USE_CONTEXT process a package from a use context clause, which
-- means it must have been withed by a prior with statement
--
-- when we enter this routine we have a package name from a context
-- clause use.  The package name must be one that was mentioned in the
-- with clause or else we print an error.  If it hasn't been used by this
-- schema before add it to the chain

------------------------------------------------------------------
--
-- PROCESS_USE_NON_CONTEXT process a package from a use non context clause,
-- which means it may be a qualified package name or it may be a subpackage
-- name from a package that has already been withed and used
--
-- when we enter this routine we have a package name from a non context
-- clause use.  The package name may be qualified with a preceding package
-- name.  But two levels is the max.  The first may be anything, the second
-- if there must be ADA_SQL.  Split the use package name into outter name
-- and inner name.  This package must then be found in a with descriptor for
-- the current schema.  If it's valid and it hasn't been used by this
```

```
-- schema before add it to the chain.  If it's invalid tell the user we can't
-- find it in a withed schema or it ambiguous.

------------------------------------------------------------------
--
-- VALID_USE - make sure the package being used is valid and has been withed
--
-- given an outter package name and/or an inner package name and a schema unit
-- descriptor find out if these package names are valid for a use clause.
--
-- We read the withed schemas for the current schema
-- if we have an outter package and it does match but we don't have an inner,
-- or we do have an inner and it matches too, count it as a match
-- if we don't have an outter but the inner matches and this withed
--    outter package was used in our schema, count it as a match, and save
--    the outter name for later

-- first determine if we have an inner package or outter package or both or
-- neither - if neither it's an error

-- loop thru all the packages withed by this schema unit and check for matches
-- if the first declared package of a schema unit matches the outter package
--    we match on outter
-- if the next declared package of the schema unit matches the inner package
--    we match on inner

-- if we have an outter and an inner and both match, that counts as a match
-- if we have an outter and it matches and we have no inner, that counts as
--    a match

-- if we don't have an outter but the inner matches we check to see if the
--    outter was previously used by this schema.  If so that counts as a
--    match and we hang on to the outter name for later use

-- if we matched one and only one package from a withed unit it's valid
-- if we're missing the outter package we stuff it into the holder
```

3.2.12 File DDLVRBLS.ADA

```
-- DDL_VARIABLES variables used during the processing of schema units
```

3.2.13 File DDLFUNCS.ADA

```
-- FUNCTION_ROUTINES process the "function x is new authorization identifier;"
```

```
-- statement

-- PROCESS_FUNCTION process the "function x is new authorization identifier;"
-- statement
```

3.2.14 File DDLFUNCB.ADA

```
-- FUNCTION_ROUTINES process the "function x is new authorization identifier;"
-- statement

------------------------------------------------------------------
--
-- PROCESS_FUNCTION process the "function x is new authorization identifier;"
-- statement
--
-- on input temp string is function, it must be followed by an identifier
-- and then "is new authorization_identifier;"  If it isn't it's invalid and
-- we don't accept an authorization identifier.  If it is valid and an
-- authorization identifier has not already been declared in this schema unit
-- then this is it and set the flag that this is the auth package.  If one has
-- already been declared in this schema unit then it's an error.  If anything
-- in the with or use other than SCHEMA_DEFINITION that's an error.
-- One package must be open and none closed or it's an error.  If we've
-- declared types or tables or variables it's an error.  If it contains the
-- suffix _NOT_NULL or _NOT_NULL_UNIQUE it's an error and if it's more than
-- 18 characters long its an error
```

3.2.15 File DDLAUTHB.ADA

```
-- SCHEMA_AUTHORIZATION_ROUTINES process the authorization clause

------------------------------------------------------------------
--
-- PROCESS_SCHEMA_AUTHORIZATION process the schema authorization clause which
-- should read "SCHEMA_AUTHORIZATION : IDENTIFIER := identifier;"
--
-- on entry temp string is schema_authorization, it should be followed by
-- ": identifier :=" and the identifier.  It must be declared in an ADA_SQL
-- sub package and match the authorization identifier from an already
-- defined authorization package that was withed.  If types or tables have
-- already been declared warn the user that the schema authorizathion should
-- come first.  If variables have been declared tell them it's an error.
```

3.2.16 File DDLAUTHS.ADA

```
-- SCHEMA_AUTHORIZATION_ROUTINES process the authorization clause

-- PROCESS_SCHEMA_AUTHORIZATION process the schema authorization clause which
-- should read "SCHEMA_AUTHORIZATION : IDENTIFIER := identifier;"
```

3.2.17 File DDLPACKS.ADA

```
-- PACKAGE_ROUTINES process a package declaration

-- PROCESS_PACKAGE process a package statement which is "PACKAGE x IS"
```

3.2.18 File DDLPACKB.ADA

```
-- PACKAGE_ROUTINES process a package declaration

------------------------------------------------------------
--
-- PROCESS_PACKAGE process a package statement which is "PACKAGE x IS"
--
-- the token we get in temp string is "package" toss it, then read the
-- identifier and set the pointers.  If this is the first package declared
-- by the schema it may be anything but ADA_SQL.  If it is the second it
-- must be ADA_SQL.  If it is third or more we'll stuff it in the chain
-- no matter what it is but it's invalid.  Tell them it's invalid if it has
-- the suffix _NOT_NULL or _NOT_NULL_UNIQUE. Gobble up the "is" after the
-- identifier too
```

3.2.19 File DDLENDS.ADA

```
-- END_ROUTINES process an end of package statement
-- PROCESS_END process an end of package statement for either the last declared
-- package or for a named package
-- END_LAST_PACKAGE process an end package statement for the last declared
-- package
-- END_NAMED_PACKAGE process an end package statement for the named package
```

3.2.20 File DDLENDB.ADA

```
-- END_ROUTINES process an end of package statement

-------------------------------------------------------------------
--
-- PROCESS_END process an end of package statement for either the last declared
-- package or for a named package
--
-- the only end we'll get here is the end of a package, it may be followed
-- by the package name or it may be followed by just a semicolon.  If a
-- package name then it better be the last defined not yet ended since
-- if there is more than one it would have to be nested.  If it's not the
-- last one but is a match tell em out of order end but go ahead and flag
-- it as done anyway.  If it's a semi colon then it matches up to the
-- lastest one not ended.  After it's processed, call set up our package name
-- to alter current package name.

-------------------------------------------------------------------
--
-- END_LAST_PACKAGE process an end package statement for the last declared
-- package
--
-- we have the end for the last unended package, the only error is if there
-- is no package to end

-------------------------------------------------------------------
--
-- END_NAMED_PACKAGE process an end package statement for the named package
--
-- we have the end for a named package, the only error is if there
-- is no package to end, or if the end is out of order since packages should
-- be nested
```

3.2.21 File DDLTYPES.ADA

```
-- TYPE_ROUTINES process a type declaration

-- PROCESS_A_TYPE process a type declaration for an array (character string),
-- integer, floating point or derived type
```

3.2.22 File DDLTYPEB.ADA

```
-- TYPE_ROUTINES process a type declaration

-------------------------------------------------------------------
--
-- PROCESS_A_TYPE process a type declaration for an array (character string),
-- integer, floating point or derived type
--
-- first thing to do is store away the identifier or identifiers
-- then find out what type we're processing, array, integer, real or derived
-- then process accrodingly by calling the appropriate routine

-- first check to determine that a type declaration is permitted here

-- then make a chain of all identifiers - return with "is" in temp_string

-- then determine if it's a type we deal with and if so call the routine
```

3.2.23 File DDLSUBS.ADA

```
-- SUBTYPE_ROUTINES process a subtype declaration

-- PROCESS_SUBTYPE process a subtype declaration of a previously declared
-- type

-- DO_A_SUBTYPE process a subtype indicator

-- BUILD_SUBTYPE_TYPE_DESCRIPTORS create a type descriptor for this subtype
```

3.2.24 File DDLSUBB.ADA

```
   SUBTYPE_ROUTINES process a subtype declaration

-------------------------------------------------------------------
--
-- PROCESS_SUBTYPE process a subtype declaration of a previously declared
-- type

-- first check to make sure a subtype declaration is valid here

-- then make a chain of all identifiers - return with "is" in temp_string

-- then process the subtype indicator and build it all into a type descriptor

-------------------------------------------------------------------
```

```
--
-- DO_A_SUBTYPE process a subtype indicator
--
-- on entry "is" is in temp_string
-- we have to process the subtype indicator, see if it's valid and add
-- a subtype type descriptor

------------------------------------------------------------------------
--
-- BUILD_SUBTYPE_TYPE_DESCRIPTORS create a type descriptor for this subtype
--
```

3.2.25 File DDLRECS.ADA

```
-- RECORD_ROUTINES process a record declaration

-- PROCESS_RECORD process a record declaration which must be the description
-- of a database table when appearing in the ddl

-- BUILD_COMPONENT_TYPE_DESCRIPTORS build the type descriptor for a component
-- of a record which is a column in a database table

-- BUILD_RECORD_TYPE_DESCRIPTORS build the type descriptor for a record which
-- is a database table

-- INSERT_COMPONENT_DESCRIPTORS stuff into a chain in the record type descriptor
-- pointers to all of it's component type descriptors
```

3.2.26 File DDLRECB.ADA

```
-- RECORD_ROUTINES process a record declaration

------------------------------------------------------------------------
--
-- PROCESS_RECORD process a record declaration which must be the description
-- of a database table when appearing in the ddl
--
-- on entry "record" is in temp_string
-- we have to process each component statement and determine if it's valid
-- read token to get first component name or "end", if end we're done with
-- the whole record, if component name call make_list_of_components to
-- stack up the component names since there may be more than one for each
-- component statement.
```

```
-- determine that the declaration o' a record (database table) is valid here

-- for each component declaration (database column)

-- stack up the identifier names since several components could be declared
-- in the same statement

-- break down and validate the subtype indicator for the component

------------------------------------------------------------------------
--
-- BUILD_COMPONENT_TYPE_DESCRIPTORS build the type descriptor for a component
-- of a record which is a column in a database table
--

------------------------------------------------------------------------
--
-- BUILD_RECORD_TYPE_DESCRIPTORS build the type descriptor for a record which
-- is a database table
--

------------------------------------------------------------------------
--
-- INSERT_COMPONENT_DESCRIPTORS stuff into a chain in the record type descriptor
-- pointers to all of it's component type descriptors
--
```

3.2.27 File DDLVARS.ADA

```
-- VARIABLE_ROUTINES process a variable declaration

-- TRY_TO_PROCESS_VARIABLE all statements which begin with an identifier
-- are processed thru this routine, try to process the identifier as
-- a variable to see if it's valid

-- PROCESS_VARIABLE process a variable subtype indicator, the identifier
-- of the variable has already been stored, and create the type descriptors

-- BUILD_VARIABLE_TYPE_DESCRIPTORS build a type descriptor for a variable
```

3.2.28 File DDLVARB.ADA

```
-- VARIABLE_ROUTINES process a variable declaration
```

```
-------------------------------------------------------------------
--
-- TRY_TO_PROCESS_VARIABLE all statements which begin with an identifier
-- are processed thru this routine, try to process the identifier as
-- a variable to see if it's valid
--
-- first thing to do is store away the identifier or identifiers
-- if there are identifiers and then a : we assume variables, otherwise
-- we assume it's a statement we know nothing about
-- then process the subtype indicator
-- then build it all into a variable descriptor

-- first make a chain of all identifiers - returns with ":" in temp_string
-- and make sure a variable declaration would be valid at this time

-- if all is valid up to this point then call the routine to process a variable

-------------------------------------------------------------------
--
-- PROCESS_VARIABLE process a variable subtype indicator, the identifier
-- of the variable has already been stored, and create the type descriptors
--
-- on entry ":" is in temp_string
-- we have to process the subtype indicator, see if it's valid and add
-- a variable type descriptor

-------------------------------------------------------------------------
--
-- BUILD_VARIABLE_TYPE_DESCRIPTORS build a type descriptor for a variable
--
```

3.2.29 File DDLINTS.ADA

```
-- INTEGER_ROUTINES process the integer section of a type declaration

-- PROCESS_INTEGER process the section of a type declaration that indicates
-- an integer declaration, "range x .. z;"

-- GET_INTEGER_RANGE read the range declaration of the statement and
-- determine if it's valid and return the high and low range

-- BUILD_INTEGER_TYPE_DESCRIPTORS build the type descriptor for the integer
-- declaration here
```

3.2.30 File DDLINTB.ADA

```
-- INTEGER_ROUTINES process the integer section of a type declaration

-------------------------------------------------------------------
--
-- PROCESS_INTEGER process the section of a type declaration that indicates
-- an integer declaration, "range x .. z;"
--
-- on entry "range" is in temp_string
-- we have to process the statement and determine if it's valid
-- the next token should be an integer for index range lo
-- followed by .. and then an integer for index range hi and then a semi colon

-- validate it and store necessary info to build the descriptor

-- build type descriptors here

-------------------------------------------------------------------------
--
-- GET_INTEGER_RANGE read the range declaration of the statement and
-- determine if it's valid and return the high and low range
--
-- if valid is false on entry then don't do anything
-- we have to find a range or valid becomes false
-- lo and hi range become the range specified,

-------------------------------------------------------------------------
--
-- BUILD_INTEGER_TYPE_DESCRIPTORS build the type descriptor for the integer
-- declaration here
--
```

3.2.31 File DDLFLTS.ADA

```
-- FLOAT_ROUTINES process the floating point section of a type declaration

--
-- PROCESS_FLOAT process the section of a type declaration that indicates
-- a floating point declaration, "digits x range z .. y"

-- GET_FLOAT_DIGITS read the digits number and make sure it's valid

-- GET_FLOAT_RANGE read the range declaration of the statement and
-- determine if it's valid and return the high and low range
```

```
-- BUILD_FLOAT_TYPE_DESCRIPTORS build the type descriptor for the floating point
-- declaration here
```

3.2.32 File DDLFLTB.ADA

```
-- FLOAT_ROUTINES process the floating point section of a type declaration

------------------------------------------------------------------
--
-- PROCESS_FLOAT process the section of a type declaration that indicates
-- a floating point declaration, "digits x range z .. y"
--
-- on entry "digits" is in temp_string
-- we have to process the statement and determine if it's valid
-- the next token must be a positive integer for digits
-- followed by either RANGE or ; -- if RANGE then
-- the next token must be a floating point number for index range lo
-- followed by .. and then a floating point for index range hi and then
-- a semi colon

-- validate it and store necessary info to build the type descriptor later

-- build type descriptors here

-------------------------------------------------------------------------
--
-- GET_FLOAT_DIGITS read the digits number and make sure it's valid
--
-- if valid is false on entry then don't do anything
-- we have to find the float digits which must be a positive integer

-------------------------------------------------------------------------
--
-- GET_FLOAT_RANGE read the range declaration of the statement and
-- determine if it's valid and return the high and low range
--
-- if valid is false on entry then don't do anything
-- we have to find a range or valid becomes false
-- lo and hi range become the range specified,

-------------------------------------------------------------------------
--
-- BUILD_FLOAT_TYPE_DESCRIPTORS build the type descriptor for the floating point
-- declaration here
--
```

3.2.33 File DDLENUMS.ADA

```
-- ENUMERATION_ROUTINES process the enumeration section of a type declaration

-- PROCESS_ENUMERATION process the section of a type declaration that indicates
-- an enumeration declaration, "(1, 1, 1);"

-- GET_ENUMERATION_LITERAL read one enumeration literal and make sure it's valid

-- VALID_ENUMERATION_LITERAL validate a string to be an enumeration literal

-- DUPLICATE_ENUMERATION_LITERAL check to see if this enumeration literal has
-- been used before in this enumeration declaration

-- BUILD_ENUMERATION_TYPE_DESCRIPTORS build the type descriptor for the
-- enumeration declaration here

-- BUILD_ENUMERATION_LITERAL_DESCRIPTORS add the enumeration literal on to the
-- chain of literals
```

3.2.34 File DDLENUMB.ADA

```
-- ENUMERATION_ROUTINES process the enumeration section of a type declaration

------------------------------------------------------------------
--
-- PROCESS_ENUMERATION process the section of a type declaration that indicates
-- an enumeration declaration, "(1, 1, 1);"
--
-- on entry "(" is in temp_string
-- we have to process the statement and determine if it's valid
-- we read enumeration literals up to the next ) or ;

-- read an enumeration literal and validate it and store the necessary info
-- to build a descriptor of it later

-- build type descriptors here

------------------------------------------------------------------------
--
-- GET_ENUMERATION_LITERAL read one enumeration literal and make sure it's valid
--
-- enumeration literals my be an identifier or a single character in a quote
-- if the first character read is a quote read until another quote
-- if the second is a quote then read for another quote
```

```
------------------------------------------------------------------------
--
-- VALID_ENUMERATION_LITERAL validate a string to be an enumeration literal
--
-- valid enumeration literals are either valid identifiers or a single
-- character between single quotes

------------------------------------------------------------------------
--
-- DUPLICATE_ENUMERATION_LITERAL check to see if this enumeration literal has
-- been used before in this enumeration declaration

------------------------------------------------------------------------
--
-- BUILD_ENUMERATION_TYPE_DESCRIPTORS build the type descriptor for the
-- enumeration declaration here
--

------------------------------------------------------------------------
--
-- BUILD_ENUMERATION_LITERAL_DESCRIPTORS add the enumeration literal on to the
-- chain of literals
--
```

3.2.35 File DDLARAYS.ADA

```
-- ARRAY_ROUTINES process the array section of a type declaration

-- PROCESS_ARRAY process the section of a type declaration that indicates
-- an array declaration, either unconstrained or constrained

-- GET_ARRAY_INDEX_TYPE read the temp string and return the index type and
-- default index range information

-- GET_ARRAY_INDEX_RANGE read the temp string to determine the range of
-- the array index

-- GET_ARRAY_TYPE_OF read the temp string to determine the type of the array
-- components

-- BUILD_STRING_TYPE_DESCRIPTORS build the type descriptor for the arrays here
```

3.2.36 File DDLARAYB.ADA

```
-- ARRAY_ROUTINES process the array section of a type declaration

------------------------------------------------------------------
--
-- PROCESS_ARRAY process the section of a type declaration that indicates
-- an array declaration, either unconstrained or constrained
--
-- on entry "array" is in temp_string
-- we have to process the statement and determine if it's valid
-- an unconstrained array is valid as follows:
--    ( index-type RANGE <> ) OF identifier
-- a constrained array is valid as follows:
--    ( index_type ) OF identifier
--    ( index_type RANGE l..h ) OF identifier
--    ( l..h ) OF identifier
-- if valid we collect the following information about the array to be stored
-- in the type descriptor:
--
-- identifier name    - to create a new identifier descriptor or be included
--                      in an existing one (captured by process_type, stored
--                      in make_list_of_names)
-- full name pointer  - a pointer to a full name descriptor pointed to from
--                      the identifier descriptor
-- string length      - hi range - lo range + 1, unless it's constrained then
--                      use zero for now
-- index type         - a pointer to the type descriptor of the index type,
--                      which must be base type of integer, if one is
--                      specified, if not we use standard.integer as the type
-- array type         - a pointer to the type descriptor of the array type,
--                      which must be a base type of character
-- constrained        - true if it is, false if it isn't
-- index range min    - if index type is supplied we have the minimum possible
--                      for the range, must be >= 0
-- index range max    - if index type is supplied we have the maximum possible
--                      for the range, must be >= 0
-- index range lo     - if an actual range is supplied this is the lo value,
--                      must be >= 0, unless the array is unconstrained then
--                      it will be -1
-- index range hi     - if an actual range is supplied this is the hi value,
--                      must be >= 0, unless the array is unconstrained then
--                      it will be -1

-- we validate the various components here and store necessary info to build
-- a type descriptor later

-- our first character must be (

-- if an index type is given it must be a base type of integer, if it's not
-- given we use standard.integer as the index type
```

```
-- next check to see if a RANGE is supplied

-- and if so get it's low and hi limits

-- now we need a )

-- and now OF

-- read the array type, it must be a base type of character

-- and at the end of the line we should have a ;

-- if there was an error we print a message and skip this declaration

-- build type descriptors here

------------------------------------------------------------------
--
-- GET_ARRAY_INDEX_TYPE read the temp string and return the index type and
-- default index range information
--
-- valid - if false on entry then don't do anything, don't alter
--         return false if we identify an attempt to define an array index
--            type and it's invalid.
--         do not alter if it's valid
--         We treat it as if we've found an identifier if it's alpha.
--         it must be a base type of integer and visible from our current
--            schema
--         if no identifier is found we use standard.integer as a default
-- got index type - true if we get one even if its the default
-- index type - identifier of the index type
-- index type last  - it's length.
-- range min - lo range from index type -1 if any integer is valid
-- range max - hi range from index type -1 if any integer is valid
-- index type des - pointer to type descriptor of index type, null if not here

------------------------------------------------------------------
--
-- GET_ARRAY_INDEX_RANGE read the temp string to determine the range of
-- the array index
--
-- if valid is false on entry then don't do anything
-- if need range then we have to find one or valid becomes false
-- set got range if we do find one
-- lo and hi range become the range specified, if got index type
-- is true then array lo and hi range better fall within the ranges on input,
-- if not valid = false.  If the range is <> then it's unconstrained and
-- we set the flag unconstrained as well as lo and hi to -1
```

```
-------------------------------------------------------------------------
--
-- GET_ARRAY_TYPE_OF read the temp string to determine the type of the array
-- components
--
-- if valid is false return
-- got_array_type = true if we indeed have one
-- array_type will be the qualified identifier name of length array_type_last
-- array_type_des if the type descriptor
-- to be valid the array type identifier must be visible

-------------------------------------------------------------------------
--
-- BUILD_STRING_TYPE_DESCRIPTORS build the type descriptor for the arrays here
--
```

3.2.37 File DDLDERS.ADA

```
-- DERIVED_ROUTINES process the derived section of a type declaration

-- PROCESS_DERIVED process the section of a type declaration that indicates
-- a derived declaration, which would be NEW subtype_indicator

-- BUILD_DERIVED_TYPE_DESCRIPTORS build the type descriptor for the derived
-- type here
```

3.2.38 File DDLDERB.ADA

```
-- DERIVED_ROUTINES process the derived section of a type declaration

-------------------------------------------------------------------
--
-- PROCESS_DERIVED process the section of a type declaration that indicates
-- a derived declaration, which would be NEW subtype_indicator
--
-- on entry "new" is in temp_string
-- we have to process the subtype indicator, see if it's valid and add
-- a derived type descriptor

-------------------------------------- ------------------------------
--
-- BUILD_DERIVED_TYPE_DESCRIPTORS build the type descriptor for the derived
-- type here
--
```

3.2.39 File DDLCALLS.ADA

```
-- CALL_TO_DDL_ROUTINES routines to initiate the ddl reader to process the
-- DDL for an application scanner DML module

-- CALL_TO_DDL_OPEN_SCHEMA_UNIT - request the ddl reader to set up an
-- environment to process selected sections of the with and use clauses
-- of a schema unit which is being processed as a dml module

-- CALL_TO_DDL_WITH request the ddl reader to process the given name of a
-- schema unit as though it were a with clause for the schema unit that was
-- identified in CALL_TO_DDL_OPEN_SCHEMA_UNIT

-- CALL_TO_DDL_USE request the ddl reader to process the given name of a
-- package as though it were a use clause for the schema unit that was
-- identified in CALL_TO_DDL_OPEN_SCHEMA_UNIT

-- CALL_TO_DDL_CLOSE terminate processing of the schema unit that was
-- identified in CALL_TO_DDL_OPEN_SCHEMA_UNIT
```

3.2.40 File DDLCALLB.ADA

```
-- CALL_TO_DDL_ROUTINES routines to initiate the ddl reader to process the
-- DDL for an application scanner DML module

------------------------------------------------------------------
--
-- CALL_TO_DDL_OPEN_SCHEMA_UNIT - request the ddl reader to set up an
-- environment to process selected sections of the with and use clauses
-- of a schema unit which is being processed as a dml module

-------------------------------------------------------------------------
--
-- CALL_TO_DDL_WITH request the ddl reader to process the given name of a
-- schema unit as though it were a with clause for the schema unit that was
-- identified in CALL_TO_DDL_OPEN_SCHEMA_UNIT

-------------------------------------------------------------------------
--
-- CALL_TO_DDL_USE request the ddl reader to process the given name of a
-- package as though it were a use clause for the schema unit that was
-- identified in CALL_TO_DDL_OPEN_SCHEMA_UNIT

-------------------------------------------------------------------------
--
-- CALL_TO_DDL_CLOSE terminate processing of the schema unit that was
-- identified in CALL_TO_DDL_OPEN_SCHEMA_UNIT
```

3.2.41 File DDLMAIN.ADA

```
-- MAIN for testing purposes this will drive the ddl reader (without adding
-- all the other application scanner code in) with input from the terminal
-- and will display all data structures created
```

3.2.42 File DDLMAINC.ADA

```
-- MAIN_CALL for testing purposes this will drive the ddl reader (without adding
-- all the other application scanner code in) in the same manner that it will
-- be called when the application scanner is executing
```

3.2.43 File DDLSIOS.ADA

```
-- SCHEMA_IO the io routines related to the schema units to open and close
-- files, to read data from files and the terminal, to output data to files
-- and the terminal, and to perform data conversions

-- OPEN_SCHEMA_UNIT open a schema unit file for processing

-- GET_STRING return the next token from the schema unit currently being
-- processed

-- CLOSE_SCHEMA_UNIT close the schema unit file currently being processed

-- PRINT ERROR print an error describing the schema unit from which the error
-- resulted and the line number

-- PRINT TO FILE print a message to the output file

-- PRINT MESSAGE print a message to the current output device, most likely
--  the terminal

-- GET_TERMINAL_INPUT obtain input from the current input device, most likely
-- the terminal

-- OPEN_OUTPUT_FILE set flags showing that the output/error file is open

-- CLOSE_OUTPUT_FILE set the flage indigating that the output file is closed

-- UPPER_CASE convert a string to upper case

-- LOWER_CASE convert a string to lower case
```

```
-- DOUBLE_PRECISION_TO_STRING return a string representation of a double
-- precision number

-- STRING_TO_DOUBLE_PRECISION return the double precision equivalent of a
-- string representing a number

-- EXCHANGE_FOR_ORIGINAL given the schema and the buffer we're working with,
-- exchange the current token which was converted to upper case on input for
-- the originally cased token

-- GET_SINGLE_QUOTE_STRING return a quoted single character from the input
-- buffer
```

3.2.44 File DDLSIOB.ADA

```
-- SCHEMA_IO the io routines related to the schema units to open and close
-- files, to read data from files and the terminal, to output data to files
-- and the terminal, and to perform data conversions

------------------------------------------------------------------------
--
-- OPEN_SCHEMA_UNIT open a schema unit file for processing
--
-- if the file is not and has not been processed then set the file name up to
-- be the library unit plus the extention of .ADA or .A or what ever is
-- defined in ddliodefs as being the extention of the system.  The case of the
-- file is determined by the flags governing case in ddliodefs.  The case of the
-- extention is determined by the case of the first letter of the file name
-- and the flags governing case in ddliodefs.  If the schema to be processed
-- is one of the special standard ones, use the correct name and directory
-- location from ddliodefs to locate the version that we should be reading.
-- Open the file if this schema was not initiated from the CALLS_TO_DDL.
-- If it was then we don't open it but just pretend to do so.  Set the status
-- to processing.  If we get an exception on opening the file print the
-- appropriate message and set status to not found.

-- reading unopen file, opening open file
-- read output or write input
-- can't find file
-- can't perform requested operation
-- device malfunction
-- eof
-- bad data
-- page format error

------------------------------------------------------------------------
--
-- GET_STRING return the next token from the schema unit currently being
```

```
-- processed
--
-- if we are not actually reading the schema unit but obtaining data thru the
-- CALL_TO_DDL routines then the underlying routines will set up our buffers
-- as if they were input from a file.

------------------------------------------------------------------------
--
-- CLOSE_SCHEMA_UNIT close the schema unit file currently being processed
--
-- remember that if this schema unit was initiated via an open call in
-- the CALL_TO_DDL routines we did not really open it but relied on input
-- from other calls from CALL_TO_DDL and we must not really close it but
-- we must set up the flags as though we did
--

-- reading unopen file, opening open file
-- read output or write input
-- can't find file
-- can't perform requested operation
-- device malfunction
-- eof
-- bad data
-- page format error

------------------------------------------------------------------------
--
-- PRINT ERROR print an error describing the schema unit from which the error
-- resulted and the line number

------------------------------------------------------------------------
--
-- PRINT TO FILE print a message to the output file
--
-- take note here that the output_file_type will create a ddl reader output
-- only (with no additional application scanner information in it) with
-- the file name of first schema unit processed and an extention of .ddlout
-- as defined in ddliodefs.  This has been changed to output all messages to
-- the same file that the remainder of the application scanner is using with
-- the call to lexical_analyzer.report_ddl_error

-- reading unopen file, opening open file
-- read output or write input
-- can't find file
-- can't perform requested operation
-- device malfunction
-- eof
-- bad data
-- page format error
```

```
------------------------------------------------------------------------
--
-- PRINT MESSAGE print a message to the current output device, most likely
--  the terminal

-- reading unopen file, opening open file
-- read output or write input
-- can't find file
-- can't perform requested operation
-- device malfunction
-- eof
-- bad data
-- page format error

------------------------------------------------------------------------
--
-- GET_TERMINAL_INPUT obtain input from the current input device, most likely
-- the terminal

-- reading unopen file, opening open file
-- read output or write input
-- can't find file
-- can't perform requested operation
-- device malfunction
-- eof
-- bad data
-- page format error

------------------------------------------------------------------------
--
-- OPEN_OUTPUT_FILE set flags showing that the output/error file is open
--
-- this routine used to open the output file to which the ddl reader would
-- output errors and information, however this file has now been merged with
-- the one used by the rest of the application scanner so the actual opening
-- of a file is not done here but the flags are set up to show that it was done.
-- If the file isn't really open this routine will not detect it.
--
-- take note here that the output_file_type will create a ddl reader output
-- only (with no additional application scanner information in it) with
-- the file name of first schema unit processed and an extention of .ddlout
-- as defined in ddliodefs.  This has been changed to output all messages to
-- the same file that the remainder of the application scanner is using with
-- the call to lexical_analyzer.report_ddl_error

-- reading unopen file, opening open file
-- can't find file
-- can't perform requested operation
-- device malfunction
```

```
-- eof
-- bad data
-- page format error

------------------------------------------------------------------------
--
-- CLOSE_OUTPUT_FILE set the flage indigating that the output file is closed
--
-- take note here that the output_file_type will create a ddl reader output
-- only (with no additional application scanner information in it) with
-- the file name of first schema unit processed and an extention of .ddlout
-- as defined in ddliodefs.  This has been changed to output all messages to
-- the same file that the remainder of the application scanner is using with
-- the call to lexical_analyzer.report_ddl_error.  Therefore we don't really
-- close a file here but just pretend to.

-- reading unopen file, opening open file
-- read output or write input
-- can't find file
-- can't perform requested operation
-- device malfunction
-- eof
-- bad data
-- page format error

------------------------------------------------------------------------
--
-- UPPER_CASE convert a string to upper case

------------------------------------------------------------------------
--
-- LOWER_CASE convert a string to lower case

------------------------------------------------------------------------
--
-- DOUBLE_PRECISION_TO_STRING return a string representation of a double
-- precision number

------------------------------------------------------------------------
--
-- STRING_TO_DOUBLE_PRECISION return the double precision equivalent of a
-- string representing a number

------------------------------------------------------------------------
--
-- EXCHANGE_FOR_ORIGINAL given the schema and the buffer we're working with,
-- exchange the current token which was converted to upper case on input for
-- the originally cased token
--
```

```
-- this routine is used when we what to know the actual case a user entered a
-- file name in - for most purposes we use all upper case thru the ddl reader
-- to avoid confusion

------------------------------------------------------------------------
--
-- GET_SINGLE_QUOTE_STRING return a quoted single character from the input
-- buffer
--
-- on entry buf_len = 1 and buf = single quote.  Keep reading till ending quote
-- however if second character is quote and third character is quote return
-- the three. Valid is true if on return buf_len = 3 and buf(1) and buf(3) = '
-- the quoted string must be all on one line or it's an error
```

3.2.45 File DDLIOINS.ADA

```
-- IO_INTERNAL_STUFF these are the routines used by SCHEMA_IO to do the
-- nitty grittys for the io routines

-- TOKEN_END bump the schema buffer pointers to the beginning of the next
-- token and return a pointer to the end of that token

-- WHITESPACE return true if character is a white space

-- ALPHABETIC return true if character is alphabetic

-- SIMPLE_NUMERIC return true if character is numeric 0 - 9 or underscore

-- QUALIFIER return true if we're pointing to the second or subsquent portion
-- of a qualified expression

-- NUMERIC return true if the character is numeric 0 - 9 or underscore or
-- + or - or . or E and could be part of a numeric string based on previous
-- characters encountered in the string

-- VALID_AFTER_DECIMAL return true is character is a valid character following
-- a decimal character in a numeric string

-- NEXT_TOKEN set the pointers in the schema buffer to point to the
-- beginning of the next token

-- NEXT_LINE read the next line from the schema unit file into the buffer
```

3.2.46 File DDLIOINB.ADA

```
-- IO_INTERNAL_STUFF these are the routines used by SCHEMA_IO to do the
-- nitty grittys for the io routines

------------------------------------------------------------------------
--
-- TOKEN_END bump the schema buffer pointers to the beginning of the next
-- token and return a pointer to the end of that token
--
-- point to beginning of token to read,  there are two possible cases for us
-- to read.  One is an alpha type - this must start with A .. Z and then may
-- be followed with A..Z 0..9 _ or . No further rules apply except to the .
-- which is assumed to be qualifying something.  If the . if the first
-- character it gets returned separately.  it must be followed by A..Z
-- not any thing else.  if two dots are found in a row we return up to
-- but not including the first one
-- the other type is numeric - it starts with a + or - or 0..9 then is
-- followed by 0..9 or _ and maybe an E.  After hitting an E we have to
-- have + or - or 0..9 and then only 0..9 or _ the rest of the token

------------------------------------------------------------------------
--
-- WHITESPACE return true if character is a white space

------------------------------------------------------------------------
--
-- ALPHABETIC return true if character is alphabetic

------------------------------------------------------------------------
--
-- SIMPLE_NUMERIC return true if character is numeric 0 - 9 or underscore

------------------------------------------------------------------------
--
-- QUALIFIER return true if we're pointing to the second or subsquent portion
-- of a qualified expression
--
-- C is the character in question and if it's not a dot it certainly isn't
-- a qualifier here.  Then if the next character is A..Z it's ok

------------------------------------------------------------------------
--
-- NUMERIC return true if the character is numeric 0 - 9 or underscore or
-- + or - or . or E and could be part of a numeric string based on previous
-- characters encountered in the string
--

------------------------------------------------------------------------
```

```
--
-- VALID_AFTER_DECIMAL return true is character is a valid character following
-- a decimal character in a numeric string

------------------------------------------------------------------------------
--
-- NEXT_TOKEN set the pointers in the schema buffer to point to the
-- beginning of the next token
--
-- we want to end up pointing at the beginning of the next token, it could
-- already be there
-- if we've reached the end of the line or a comment, read the next line
-- skip leading spaces and horizontal tabs

------------------------------------------------------------------------------
--
-- NEXT_LINE read the next line from the schema unit file into the buffer
--
-- we read a line from the file if it's really ready to be processed
-- don't keep comment lines
-- if we get an exception - we're expecting eof sooner or later - we print
-- a message if anything other than eof and set SCHEMA.SCHEMA_STATUS to
-- DONE and close the file
-- and set schema.stream.buffer(1..2) to spaces and schema.stream.next
-- to 1 and schema.stream.last to 1.

-- reading unopen file, opening open file
-- read output or write input
-- can't find file
-- can't perform requested operation
-- device malfunction
-- eof
-- bad data
-- page format error
```

3.2.47 File DDLIOERS.ADA

```
-- IO_ERRORS these are the error routines used by SCHEMA_IO for the io routines

-- OPEN_ERROR got an exception while trying to open a schema unit

-- READ_ERROR got an exception while reading from a schema unit file

-- CLOSE_ERROR got an exception when trying to close a schema unit file

-- PRINT_ERROR_ERROR got an exception while trying to write to the output file

-- PRINT_MESSAGE_ERROR got an exception while trying to write to the terminal
```

```
-- INPUT_ERROR got an exception while trying to read from terminal

-- OPEN_OUTPUT_FILE_ERROR got an exception when trying to open the output file

-- CLOSE_OUTPUT_FILE_ERROR got an exception when trying to close the output file
```

3.2.48 File DDLIOERB.ADA

```
-- IO_ERRORS these are the error routines used by SCHEMA_IO for the io routines

------------------------------------------------------------------------
--
-- OPEN_ERROR got an exception while trying to open a schema unit

------------------------------------------------------------------------
--
-- READ_ERROR got an exception while reading from a schema unit file
--
-- we got an exception while reading - we're expecting eof sooner or later -
-- we print the message if anything other than eof
-- set SCHEMA.SCHEMA_STATUS to DONE
-- set schema.stream.buffer(1..2) to spaces
-- schema.stream.next to 1
-- schema.stream.last to 1.
-- close the file

------------------------------------------------------------------------
--
-- CLOSE_ERROR got an exception when trying to close a schema unit file

------------------------------------------------------------------------
--
-- PRINT_ERROR_ERROR got an exception while trying to write to the output file

------------------------------------------------------------------------
--
-- PRINT_MESSAGE_ERROR got an exception while trying to write to the terminal

------------------------------------------------------------------------
--
-- INPUT_ERROR got an exception while trying to read from terminal

------------------------------------------------------------------------
--
-- OPEN_OUTPUT_FILE_ERROR got an exception when trying to open the output file

------------------------------------------------------------------------
```

```
--
-- CLOSE_OUTPUT_FILE_ERROR got an exception when trying to close the output file
```

3.2.49 File DDLADESS.ADA

```
-- ADD_DESCRIPTOR_ROUTINES add various descriptors to various chains

-- ADD-YET_TO_DO_DESCRIPTOR add a descriptor to the chain of schema units
-- that have not yet been completely processed

-- ADD_SCHEMA_UNIT_DESCRIPTOR add a new descriptor for a schema unit to
-- the chain of schema units processed

-- ADD_WITHED_UNIT_DESCRIPTOR  add a withed unit descriptor for a library
-- unit which was withed by the schema unit to the chain of withed unit
-- descriptors for within the schema unit descriptor

-- ADD_USED_PACKAGE_DESCRIPTOR add a used package descriptor for a package
-- which was used by the schema unit to the chain of used package descriptors
-- within the schema unit descriptor

-- ADD_DECLARED_PACKAGE_DESCRIPTOR add a declared package descriptor for a
-- package which was declared by the schema unit to the chain of declared
-- package descriptors within the schema unit

-- ADD_IDENTIFIER_DESCRIPTOR add a descriptor for an identifier, which has been
-- defined by a schema unit, to the identifier chain

-- ADD_FULL_NAME_DESCRIPTOR  add a full name descriptor for an identifier which
-- has been declared by a schema unit, to the full name chain, the fully
-- qualified name of that identifier will be retained and the identifier
-- descriptor will be pointed to

-- ADD_TYPE_DESCRIPTOR add a type descriptor of any one of the various types
-- to the chain of type descriptors

-- ADD_VARIABLE_TYPE_DESCRIPTOR  add a type descriptor for a variable to the
-- chain of variables

-- ADD_RECORD_TYPE_DESCRIPTOR add a descriptor of a record (database table)
-- to the chain of database tables

-- ADD_LITERAL_DESCRIPTOR add the descriptor for an enumeration literal to the
-- chain of literals within the enumeration descriptor

-- ADD_ENUM_IDENT_DESCRIPTOR add an enumeration literal descriptor to the
```

```
-- chain of all literals

-- ADD_FULL_ENUM_LIT_DESCRIPTOR add an enumeration literal descriptor to the
-- chain of all fully qualified literals which retain the fully qualified names
-- and point to the literal descriptor
```

3.2.50 File DDLADESB.ADA

```
-- ADD_DESCRIPTOR_ROUTINES add various descriptors to various chains

------------------------------------------------------------------------------
--
-- ADD-YET_TO_DO_DESCRIPTOR add a descriptor to the chain of schema units
-- that have not yet been completely processed
--
-- if this is the first yet-to-do defined set the first pointer
-- otherwise set the "next" pointer in the previously last yet-to-do to
--           point to this new yet-to-do
-- set the previous pointer in this new yet-to-do to point to the
--           old last yet-to-do
-- and now the new yet-to-do is the last one

------------------------------------------------------------------------------
--
-- ADD_SCHEMA_UNIT_DESCRIPTOR add a new descriptor for a schema unit to
-- the chain of schema units processed
--
-- if this is the first schema unit defined set the first pointer
-- otherwise set the "next" pointer in the previously last schema unit to
--           point to this new schema unit
-- set the previous pointer in this new schema unit to point to the
--           old last schema unit
-- and now the new schema unit is the last one

------------------------------------------------------------------------------
--
-- ADD_WITHED_UNIT_DESCRIPTOR  add a withed unit descriptor for a library
-- unit which was withed by the schema unit to the chain of withed unit
-- descriptors for within the schema unit descriptor
--
-- if this is the first withed unit defined for this schema unit set the
--           first pointer
-- otherwise set the "next" pointer in the previously last withed unit to
--           point to this new withed unit
-- set the previous pointer in this new withed unit to point to the
--           old last withed unit
-- and now the new withed unit is the last one pointed to by the schema
```

```
------------------------------------------------------------------------
--
-- ADD_USED_PACKAGE_DESCRIPTOR add a used package descriptor for a package
-- which was used by the schema unit to the chain of used package descriptors
-- within the schema unit descriptor
--
-- if this is the first used unit defined for this schema unit set the
--           first pointer
-- otherwise set the "next" pointer in the previously last used unit to
--           point to this new used unit
-- set the previous pointer in this new used unit to point to the
--           old last used unit
-- and now the new used unit is the last one pointed to by the schema

------------------------------------------------------------------------
--
-- ADD_DECLARED_PACKAGE_DESCRIPTOR add a declared package descriptor for a
-- package which was declared by the schema unit to the chain of declared
-- package descriptors within the schema unit
--
-- if this is the first declared package for this schema unit set the
--           first pointer
-- otherwise set the "next" pointer in the previously last declared package
--           to point to this new declared package
-- set the previous pointer in this new declared package to point to the
--           old last declared package
-- and now the new declared package is the last one pointed to by the schema

------------------------------------------------------------------------
--
-- ADD_IDENTIFIER_DESCRIPTOR add a descriptor for an identifier, which has been
-- defined by a schema unit, to the identifier chain
--
-- if this is the first declared identifier set the first pointer
-- otherwise set the "next" pointer in the previously last identifier
--           to point to this new identifier
-- set the previous pointer in this new identifier to point to the
--           old last identifier
-- and now the new identifier is the last one

------------------------------------------------------------------------
--
-- ADD_FULL_NAME_DESCRIPTOR  add a full name descriptor for an identifier which
-- has been declared by a schema unit, to the full name chain, the fully
-- qualified name of that identifier will be retained and the identifier
-- descriptor will be pointed to
--
-- if this is the first declared full name for this identifier set the first
--              pointer
```

```
-- otherwise set the "next" pointer in the previously last full name
--            to point to this new full name
-- set the previous pointer in this new full name to point to the old last full
--            name in the identifier descriptor
-- and now the new full name is the last one for this identifier

------------------------------------------------------------------------
--
-- ADD_TYPE_DESCRIPTOR add a type descriptor of any one of the various types
-- to the chain of type descriptors
--
-- if this is the first type set the first pointer
-- otherwise set the "next" pointer in the previously last type to point
--      to this new type
-- set the previous pointer in this new type to point to the old last type
-- and now the new type is the last one

------------------------------------------------------------------------
--
-- ADD_VARIABLE_TYPE_DESCRIPTOR  add a type descriptor for a variable to the
-- chain of variables
--
-- if this is the first variable set the first pointer
-- otherwise set the "next" pointer in the previously last variable to point
--      to this new variable
-- set the previous pointer in this new variable to point to the
--      old last variable
-- and now the new variable is the last one

------------------------------------------------------------------------
--
-- ADD_RECORD_TYPE_DESCRIPTOR add a descriptor of a record (database table)
-- to the chain of database tables
--
-- if this is the first table set the first pointer
-- otherwise set the "next" pointer in the previously last table to point
--      to this new table
-- set the previous pointer in this new table to point to the old last table
-- and now the new table is the last one

------------------------------------------------------------------------
--
-- ADD_LITERAL_DESCRIPTOR add the descriptor for an enumeration literal to the
-- chain of literals within the enumeration descriptor
--
-- if this is the first literal defined for this enumeration type set the
--           first pointer
-- otherwise set the "next" pointer in the previously last literal to
--           point to this new literal
```

```
-- set the previous pointer in this new literal to point to the
--          old last literal
-- and now the new literal is the last one pointed to by the enumeration type

------------------------------------------------------------------------------
--
-- ADD_ENUM_IDENT_DESCRIPTOR add an enumeration literal descriptor to the
-- chain of all literals
--
-- if this is the first enumeration literal set the first pointer
-- otherwise set the "next" pointer in the previously last enumeration literal
--          to point to this new enumeration literal
-- set the previous pointer in this new enumeration literal to point to the
--          old last enumeration literal
-- and now the new enumeration literal is the last one

------------------------------------------------------------------------------
--
-- ADD_FULL_ENUM_LIT_DESCRIPTOR add an enumeration literal descriptor to the
-- chain of all fully qualified literals which retain the fully qualified names
-- and point to the literal descriptor
--
-- if this is the first full type descriptor for this enumeration literal
--             set the first pointer
-- otherwise set the "next" pointer in the previously last full enumeration
--           literal to point to this new full enumeration literal
-- set the previous pointer in this new full enumeration literal to point to
--           the old last full enumeration literal in the chain
-- and now the new full enumeration literal is the last one for this
--           enumeration literal
```

3.2.51 File DDLKEYS.ADA

```
-- KEYWORD_ROUTINES identifies the SQL and ADA key words which cannot be used
-- as identifiers

-- SQL_KEY_WORD return true if the string is a sql key word, false if not

-- ADA_KEY_WORD return true if the string is an ada key word, false if not
```

3.2.52 File DDLKEYB.ADA

```
-- KEYWORD_ROUTINES identifies the SQL and ADA key words which cannot be used
-- as identifiers
```

```
------------------------------------------------------------------------
--
-- table of the SQL key words which cannot be used as identifiers

------------------------------------------------------------------------
--
-- table of the ADA key words which cannot be used as identifiers

------------------------------------------------------------------------
--
-- SQL_KEY_WORD return true if the string is a sql key word, false if not

------------------------------------------------------------------------
--
-- ADA_KEY_WORD return true if the string is an ada key word, false if not
```

3.2.53 File DDLLISTS.ADA

```
-- LIST_ROUTINES form the chains which hold the identifiers for type
-- variable and record component (database columns) declarations, for which
-- type descriptors will be created the remainder of the declatation statement
-- is valid

-- MAKE_LIST_OF_NAMES form a chain of identifiers from a type or subtype
-- declaration

-- ADD_NAME_TO_PROCESS_LIST  add an identifier name to the list of identifiers
-- from a type or subtype declaration that need to be processed

-- GET_NEW_LIST_NAME given a string return a list_name

-- GET_NEW_NAME_TO_PROCESS_LIST return an empty name_to_process_list

-- MAKE_LIST_OF_COMPONENTS form a chain of component identifiers (database
-- table column names) from record component declaration

-- ADD_COMPONENT_TO_PROCESS_LIST add a component name to the list of components
-- from a record declaration that need to be processed

-- GET_NEW_LIST_COMPONENT given a string return a list_component

-- GET_NEW_COMPONENT_TO_PROCESS_LIST return an empty component_to_process_list

-- MAKE_LIST_OF_VARIABLES form a chain of variable names from a variable
-- declaration
```

3.2.54 File DDLLISTB.ADA

```
-- LIST_ROUTINES form the chains which hold the identifiers for type
-- variable and record component (database columns) declarations, for which
-- type descriptors will be created the remainder of the declatation statement
-- is valid  .

-------------------------------------------------------------------
--
-- MAKE_LIST_OF_NAMES form a chain of identifiers from a type or subtype
-- declaration
--
-- the next read should point us to a name of a type, derived type or subtype
-- we want to chain up a list of them to process later
-- stop when we find IS or ;
-- temp string will contain TYPE or SUBTYPE on entry
-- identifier is invalid if TYPE declaration and suffix of _NOT_NULL or
-- _NOT_NULL_UNIQUE

-------------------------------------------------------------------------
--
-- ADD_NAME_TO_PROCESS_LIST  add an identifier name to the list of identifiers
-- from a type or subtype declaration that need to be processed
--
-- if this is the first name-to-process set the first pointer
-- otherwise set the "next" pointer in the previously last name-to-process to
--           point to this new name-to-process
-- set the previous pointer in this new name-to-process to point to the
--           old last name-to-process
-- and now the new name-to-process is the last one

-------------------------------------------------------------------------
--
-- GET_NEW_LIST_NAME given a string return a list_name
--

-------------------------------------------------------------------------
--
-- GET_NEW_NAME_TO_PROCESS_LIST return an empty name_to_process_list
--

-------------------------------------------------------------------
--
-- MAKE_LIST_OF_COMPONENTS form a chain of component identifiers (database
-- table column names) from record component declaration
--
-- on entry we should point to a component of a record type
-- we want to chain up a list of them to process later
-- stop when we find : or ;
```

```
-- temp string will contain a component name on entry
-- they must not contain _NOT_NULL or _NOT_NULL_UNIQUE suffixes and must be no
-- more than 18 characters long

------------------------------------------------------------------------
--
-- ADD_COMPONENT_TO_PROCESS_LIST add a component name to the list of components
-- from a record declaration that need to be processed
--
-- if this is the first component-to-process set the first pointer
-- otherwise set the "next" pointer in the previously last
-- component-to-process to point to this new component-to-process
-- set the previous pointer in this new component-to-process to point to the
--           old last component-to-process
-- and now the new component-to-process is the last one

------------------------------------------------------------------------
--
-- GET_NEW_LIST_COMPONENT given a string return a list_component
--

------------------------------------------------------------------------
--
-- GET_NEW_COMPONENT_TO_PROCESS_LIST return an empty component_to_process_list
--

-----------------------------------------------------------------
--
-- MAKE_LIST_OF_VARIABLES form a chain of variable names from a variable
-- declaration
--
-- on entry we should point to a variable name
-- we want to chain up a list of them to process later
-- stop when we find : or ;
-- temp string will contain a variable name on entry
-- they must not contain _NOT_NULL or _NOT_NULL_UNIQUE suffixes
-- they must be unique
```

3.2.55 File DDLNAMES.ADA

```
-- NAME_ROUTINES validate identifiers

--      eof = end of file reached
--      eol = end of line ; reached
--      eoi = end of identifiers reached
--      comma = got a comma
--      valid_ident = got a valid identifier
```

```
--      invalid_ident = got an invalid identifier

-- VALID_QUALIFIED_IDENT_CHARS validate a qualified identifier

-- VALID_NEW_TABLE_NAME validates a new table name

-- VALID_NEW_IDENT_NAME_DUPS_OK validate a new identifier name, duplicating
-- the name of an existing identifier is not an error

-- VALID_NEW_IDENT_NAME validate the name of a new identifier

-- VALID_IDENT_CHARS validate the characters within an identifier name

-- DUPLICATE_IDENT_NAME check to see if the identifier name is a duplicate

-- GOT_INVALID_CONSTRAINTS validate for the _NOT_NULL and _NOT_NULL_UNIQUE
-- suffixes

-- CHECK_EOF_EOL_IS_COMMA return a flag indicating if the string represents
-- end of file, end of a line, "is", comma or a valid identifier

--  CHECK_EOF_EOL_COLON_COMMA return a flag indicating if the string represents
-- end of file, end of a line, colon, comma or a valid identifier

-- VALID_NEW_TYPE_IDENT validate a new type identifier

-- VALID_NEW_COMPONENT_IDENT validate a new component identifier

-- VALID_NEW_PACKAGE_NAME validate a new package name

-- VALID_NEW_SUBTYPE_IDENT validate a new subtype identifier

-- VALID_NEW_FULL_COMPONENT_NAME validate a new component (database column) name

-- DUPLICATE_COMPONENT_NAME check to see if this component (database column)
-- name is a duplicate within the record (database table)

-- VALID_NEW_VARIABLE_IDENT validate a new variable identifier
```

3.2.56 File DDLNAMEB.ADA

```
-- NAME_ROUTINES validate identifiers

------------------------------------------------------------------------
--
-- VALID_QUALIFIED_IDENT_CHARS validate a qualified identifier
--
```

```
-- a valid qualified identifier may consist of only an identifier, or one or
-- two packages qualifying the identifier.  Errors are:
-- more than two package qualifiers
-- any character other than a-z 0-9 _ .
-- if a package or identifier begins with a character other than a-z

------------------------------------------------------------------------
--
-- VALID_NEW_TABLE_NAME validates a new table name
--
-- given a new table identifier validate it, for characters and to see if it's
-- already been used or if it's a keyword.  It may have been used previously
-- as an identifier with different package names, in which case if the package
-- names are visible we should print a warning message.  If there is an
-- identifier descriptor for it return it.  If there is a matching table name
-- used by another schema with the same authorization id it's invalid.  It may
-- not contain the _not_null or _not_null_unique suffix, and may be no more than
-- 18 characters long.

------------------------------------------------------------------------
--
-- VALID_NEW_IDENT_NAME_DUPS_OK validate a new identifier name, duplicating
-- the name of an existing identifier is not an error
--
-- given a string determine if it's valid characters A..Z 0..9 or _ and first
--       character A..Z
-- if the current package name isn't the standard then we cannot have names
--       the same as sql or ada keywords

------------------------------------------------------------------------
--
-- VALID_NEW_IDENT_NAME validate the name of a new identifier
--
-- given a string determine if it's valid characters A..Z 0..9 or _ and first
--       character A..Z
-- if the current package name isn't the standard then we cannot have names
--       the same as sql or ada keywords
-- then check for a duplicate name

------------------------------------------------------------------------
--
-- VALID_IDENT_CHARS validate the characters within an identifier name
--
-- return false if first character is not A..Z and remaining characters aren't
-- A..Z 0..9 or _

------------------------------------------------------------------------
--
-- DUPLICATE_IDENT_NAME check to see if the identifier name is a duplicate
```

```
--
-- if it's not in the identifier_descriptors it's looking good
-- if it is then we have to make sure that the package name in the full
-- name descriptor isn't duplicated.  if it was used previously
-- as an identifier with a different package name, then if the package
-- names are both visible print a warning message.

------------------------------------------------------------------------
--
-- GOT_INVALID_CONSTRAINTS validate for the _NOT_NULL and _NOT_NULL_UNIQUE
-- suffixes

------------------------------------------------------------------------
--
-- CHECK_EOF_EOL_IS_COMMA return a flag indicating if the string represents
-- end of file, end of a line, "is", comma or a valid identifier

------------------------------------------------------------------------
--
--  CHECK_EOF_EOL_COLON_COMMA return a flag indicating if the string represents
-- end of file, end of a line, colon, comma or a valid identifier

------------------------------------------------------------------------
--
-- VALID_NEW_TYPE_IDENT validate a new type identifier
--
-- if we've reached end of file return eof
-- if we've reached semicolon end of line return eol
-- if we've reached the IS return eoi
-- if it's a comma return comma
-- then check identifier for validity

------------------------------------------------------------------------
--
-- VALID_NEW_COMPONENT_IDENT validate a new component identifier
--
-- if we've reached end of file return eof
-- if we've reached semicolon end of line return eol
-- if we've reached the : return eoi
-- if it's a comma return comma
-- then check identifier for validity

------------------------------------------------------------------------
--
-- VALID_NEW_PACKAGE_NAME validate a new package name
--
-- If this is the first package declared
-- by the schema it may be anything but ADA_SQL.  If it is the second it
-- must be ADA_SQL.  If it is third or more we'll stuff it in the chain
```

```
-- no matter what it is but it's invalid.  Tell them it's invalid if it has
-- the suffix _NOT_NULL or _NOT_NULL_UNIQUE.

------------------------------------------------------------------
--
-- VALID_NEW_SUBTYPE_IDENT validate a new subtype identifier
--
-- if we've reached end of file return eof
-- if we've reached semicolon end of line return eol
-- if we've reached the IS return eoi
-- if it's a comma return comma
-- then check identifier for validity

-------------------------------------------------------------------------
--
-- VALID_NEW_FULL_COMPONENT_NAME validate a new component (database column) name
--
-- given a string determine if it's valid characters A..Z 0..9 or _ and first
--       character A..Z
-- if the current package name isn't the standard then we cannot have names
--       the same as sql or ada keywords
-- then check for a duplicate component name

-------------------------------------------------------------------
--
-- DUPLICATE_COMPONENT_NAME check to see if this component (database column)
-- name is a duplicate within the record (database table)
--
-- if it's not in the identifier_descriptors it's looking good
-- if it is and the table names aren't the same than we're ok
-- if it is and the table names are the same, then we have to make sure
-- that the package name in the full name descriptor isn't duplicated.
-- if it was used previously as an identifier with a different package name,
-- but the same record name, then if the package names are both visible print
-- a warning message.

-----------------------------------------------------------------
--
-- VALID_NEW_VARIABLE_IDENT validate a new variable identifier
--
-- if we've reached end of file return eof
-- if we've reached semicolon end of line return eol
-- if we've reached the : return eoi
-- if it's a comma return comma
-- then check identifier for validity
-- if it looks like an identifier but has constraints return invalid_identifier
-- if it really doesn't look like an identifier return unknown
```

3.2.57 File DDLNDESS.ADA

```
-- GET_NEW_DESCRIPTOR_ROUTINES create and initialize various elements of the
-- data structures in which the ddl reader will store data

-- GET_NEW_YET_TO_DO_DESCRIPTOR for the chain of schema units not yet complete

-- GET_NEW_SCHEMA_UNIT_DESCRIPTOR for descriptions of a schema unit

-- GET_NEW_WITHED_UNIT_DESCRIPTOR for descriptions of the library units withed
-- by a schema unit

-- GET_NEW_USED_PACKAGE_DESCRIPTOR for description of a package withed by
-- a schema unit

-- GET_NEW_DECLARED_PACKAGE_DESCRIPTOR describes a package declaration within
-- a schema unit

-- GET_NEW_IDENTIFIER_DESCRIPTOR describes an identifier

-- GET_NEW_FULL_NAME_DESCRIPTOR describes the fully qualified name of an
-- identifier

-- GET_NEW_RECORD_DESCRIPTOR describes a type declaration for a record
-- (database table)

-- GET_NEW_ENUMERATION_DESCRIPTOR description for enumeration type declaration

-- GET_NEW_INTEGER_DESCRIPTOR description for an integer type declaration

-- GET_NEW_FLOAT_DESCRIPTOR description for a float type declaration

-- GET_NEW_STRING_DESCRIPTOR describtion for a string type declaration

-- GET_NEW_TYPE_DESCRIPTOR description for a record, enumeration, integer,
-- float or string type declaration

-- GET_NEW_LITERAL_DESCRIPTOR description of an enumeration literal within
-- the enumeration declaration description

-- GET_NEW_ENUM_LIT_DESCRIPTOR description of an enumeration literal within
-- the chain of literals

-- GET_NEW_FULL_ENUM_LIT_DESCRIPTOR description of an enumeration literal within
-- the chain of literals

-- GET_NEW_ENUM_LIT_NAME convert a string to an enum_lit_name type

-- GET_NEW_AUTH_IDENT_NAME convert a string to an auth_ident_name type
```

```
-- GET_NEW_LIBRARY_UNIT_NAME convert a string to a library_unit_name type

-- GET_NEW_PACKAGE_NAME convert a string to a package_name type

-- GET_NEW_RECORD_NAME convert a string to a record_name type

-- GET_NEW_TYPE_NAME convert a string to a type_name type

-- GET_NEW_ENUMERATION_NAME convert a string to an enumeration_name type
```

3.2.58 File DDLNDESB.ADA

```
-- GET_NEW_DESCRIPTOR_ROUTINES create and initialize various elements of the
-- data structures in which the ddl reader will store data

-------------------------------------------------------------------------------
--
-- GET_NEW_YET_TO_DO_DESCRIPTOR for the chain of schema units not yet complete
--

-------------------------------------------------------------------------------
--
-- GET_NEW_SCHEMA_UNIT_DESCRIPTOR for descriptions of a schema unit
--

-------------------------------------------------------------------------------
--
-- GET_NEW_WITHED_UNIT_DESCRIPTOR for descriptions of the library units withed
-- by a schema unit
--

-------------------------------------------------------------------------------
--
-- GET_NEW_USED_PACKAGE_DESCRIPTOR for description of a package withed by
-- a schema unit
--

-------------------------------------------------------------------------------
--
-- GET_NEW_DECLARED_PACKAGE_DESCRIPTOR describes a package declaration within
-- a schema unit
--

-------------------------------------------------------------------------------
--
-- GET_NEW_IDENTIFIER_DESCRIPTOR describes an identifier
--
```

```
------------------------------------------------------------------------
--
-- GET_NEW_FULL_NAME_DESCRIPTOR describes the fully qualified name of an
-- identifier
--

------------------------------------------------------------------------
--
-- GET_NEW_RECORD_DESCRIPTOR describes a type declaration for a record
-- (database table)
--

------------------------------------------------------------------------
--
-- GET_NEW_ENUMERATION_DESCRIPTOR description for enumeration type declaration
--

------------------------------------------------------------------------
--
-- GET_NEW_INTEGER_DESCRIPTOR description for an integer type declaration
--

------------------------------------------------------------------------
--
-- GET_NEW_FLOAT_DESCRIPTOR description for a float type declaration
--

------------------------------------------------------------------------
--
-- GET_NEW_STRING_DESCRIPTOR describtion for a string type declaration
--

------------------------------------------------------------------------
--
-- GET_NEW_TYPE_DESCRIPTOR description for a record, enumeration, integer,
-- float or string type declaration
--

------------------------------------------------------------------------
--
-- GET_NEW_LITERAL_DESCRIPTOR description of an enumeration literal within
-- the enumeration declaration description
--

------------------------------------------------------------------------
--
-- GET_NEW_ENUM_LIT_DESCRIPTOR description of an enumeration literal within
-- the chain of literals
--
```

```
------------------------------------------------------------------------
--
-- GET_NEW_FULL_ENUM_LIT_DESCRIPTOR description of an enumeration literal within
-- the chain of literals
--

------------------------------------------------------------------------
--
-- GET_NEW_ENUM_LIT_NAME convert a string to an enum_lit_name type
--

------------------------------------------------------------------------
--
-- GET_NEW_AUTH_IDENT_NAME convert a string to an auth_ident_name type
--

------------------------------------------------------------------------
--
-- GET_NEW_LIBRARY_UNIT_NAME convert a string to a library_unit_name type
--

------------------------------------------------------------------------
--
-- GET_NEW_PACKAGE_NAME convert a string to a package_name type
--

------------------------------------------------------------------------
--
-- GET_NEW_RECORD_NAME convert a string to a record_name type
--

------------------------------------------------------------------------
--
-- GET_NEW_TYPE_NAME convert a string to a type_name type
--

------------------------------------------------------------------------
--
-- GET_NEW_ENUMERATION_NAME convert a string to an enumeration_name type
```

3.2.59 File DDLSDESS.ADA

```
-- SEARCH_DESCRIPTOR_ROUTINES page thru the data structures and return
-- pointers to or information about various descriptors

-- FIND_NEXT_YET_TO_DO_DESCRIPTOR return a pointer to the next schema unit
-- which we should continue to process

-- FIND_SCHEMA_UNIT_DESCRIPTOR given the name of a schema unit, return a pointer
```

```
-- to its descriptor if processing has begun on it

-- DUPLICATE_WITH given the current schema we're processing and the schema of
-- the library unit we're thinking about withing, tell us if we've withed
-- this one from this schema before

-- SEARCH_WITHS_TO_FIND_A_USE given a schema_unit_descriptor and a used
-- package name, return true if that package name is that of a withed schema,
-- false if it's not

-- DUPLICATE_USE given the current schema we're processing and the full name
-- of a used package tell us if we've used this one from this schema before

-- GET_PACKAGE_COUNT count the number of packages already declared by this
-- schema unit and the number not ended yet

-- SCHEMA_AUTHORIZATION_MATCHES_AUTHORIZATION_PACKAGE see if this
-- authorization identifier has been declared in an authorization package
-- withed by the current schema

-- SET_UP_OUR_PACKAGE_NAME set up in our_package_name the package name we're
-- in right now
```

3.2.60 File DDLSDESB.ADA

```
-- SEARCH_DESCRIPTOR_ROUTINES page thru the data structures and return
-- pointers to or information about various descriptors

------------------------------------------------------------------------
--
-- FIND_NEXT_YET_TO_DO_DESCRIPTOR return a pointer to the next schema unit
-- which we should continue to process
--
-- return a schema unit descriptor of the next one to do
-- if LAST_YET_TO_DO is null we return null and that means every thing's
--         been done
-- otherwise LAST_YET_TO_DO becomes the one we're going to do and
--         LAST_YET_TO_DO is reset with PREVIOUS_YET_TO_DO
-- and PREVIOUS_YET_TO_DO's NEXT pointer is nullified

------------------------------------------------------------------------
--
-- FIND_SCHEMA_UNIT_DESCRIPTOR given the name of a schema unit, return a pointer
-- to its descriptor if processing has begun on it
--
-- return pointer to schema unit with given library unit name, if none then
```

```
--       return null
-- it will only been found if it has been processed or partially processed

------------------------------------------------------------------------------
--
-- DUPLICATE_WITH given the current schema we're processing and the schema of
-- the library unit we're thinking about withing, tell us if we've withed
-- this one from this schema before

------------------------------------------------------------------------------
--
-- SEARCH_WITHS_TO_FIND_A_USE given a schema_unit_descriptor and a used
-- package name, return true if that package name is that of a withed schema,
-- false if it's not
--
-- this is for the case of use clause in the context where it's name must
-- match exactly that of a withed unit

------------------------------------------------------------------------------
--
-- DUPLICATE_USE given the current schema we're processing and the full name
-- of a used package tell us if we've used this one from this schema before

-------------------------------------------------------------------
--
-- GET_PACKAGE_COUNT count the number of packages already declared by this
-- schema unit and the number not ended yet

-------------------------------------------------------------------
--
-- SCHEMA_AUTHORIZATION_MATCHES_AUTHORIZATION_PACKAGE see if this
-- authorization identifier has been declared in an authorization package
-- withed by the current schema

-------------------------------------------------------------------------
--
-- SET_UP_OUR_PACKAGE_NAME set up in our_package_name the package name we're
-- in right now
```

3.2.61 File DDLSHOWS.ADA

```
-- SHOW_ROUTINES print the information collected in the data structures by
-- the ddl reader

-- SHOW_DATA display the schema units
```

```
-- SHOW_SCHEMA_UNITS display the schema units which have been processed

-- SHOW_IDENTIFIERS display the identifiers which have been processed

-- SHOW_RECORD display the information that has been collected about record
-- declaration (database table)

-- SHOW_ENUMERATION display the information collected about an enumeration
-- declaration

-- SHOW_INTEGER display the information collected about an integer declaration

-- SHOW_FLOAT display the information collected about a float declaration

-- SHOW_STRING display the information collected about a string declaration

-- SHOW_POINTERS display the values of the pointers used by the ddl reader

-- SHOW_ENUMS display the enumeration literal chain
```

3.2.62 File DDLSHOWB.ADA

```
-- SHOW_ROUTINES print the information collected in the data structures by
-- the ddl reader

------------------------------------------------------------------
--
-- SHOW_DATA display the schema units

------------------------------------------------------------------
--
-- SHOW_SCHEMA_UNITS display the schema units which have been processed

------------------------------------------------------------------
--
-- SHOW_IDENTIFIERS display the identifiers which have been processed

------------------------------------------------------------------
--
-- SHOW_RECORD display the information that has been collected about record
-- declaration (database table)

------------------------------------------------------------------
--
-- SHOW_ENUMERATION display the information collected about an enumeration
-- declaration
```

```
-------------------------------------------------------------
--
-- SHOW_INTEGER display the information collected about an integer declaration

-------------------------------------------------------------
--
-- SHOW_FLOAT display the information collected about a float declaration

-------------------------------------------------------------
--
-- SHOW_STRING display the information collected about a string declaration

-------------------------------------------------------------
--
-- SHOW_POINTERS display the values of the pointers used by the ddl reader

-------------------------------------------------------------
--
-- SHOW_ENUMS display the enumeration literal chain
```

3.2.63 File DDLERRS.ADA

```
-- ERROR_ROUTINES handel an unknown error

-- PROCESS_ERROR print a message about an error unknown to the ddl reader
```

3.2.64 File DDLERRB.ADA

```
-- ERROR_ROUTINES handel an unknown error

-------------------------------------------------------------
--
-- PROCESS_ERROR print a message about an error unknown to the ddl reader
```

3.2.65 File DDLSUB1S.ADA

```
-- SUBROUTINES_1_ROUTINES contain some of the subroutines used by the ddl reader

-- SPLIT_PACKAGE_NAME split a possibly qualified package name into an inner
```

```
-- package name and an outter package

-- FIND_END_OF_STATEMENT advance pointers to the end of the current statement

-- GOT_END_OF_STATEMENT determine if we're at the end of the current statement

-- GET_CONSTANT does the current token match this constant

-- GET_CONSTANT_MAYBE if the current token matches this constant advance
-- pointers past this tokan

-- ADJUST_USER_SCHEMA manipulate the schema name to the format we want

-- CHARACTER_STRINGS_MATCH if the two strings match regardless of case
-- return true
```

3.2.66 File DDLSUB1B.ADA

```
-- SUBROUTINES_1_ROUTINES contain some of the subroutines used by the ddl reader

------------------------------------------------------------------
--
-- SPLIT_PACKAGE_NAME split a possibly qualified package name into an inner
-- package name and an outter package
--
-- given inner package which may be two packages (inner.outter)
-- split them into two packages, if only one return as outter,
-- unless it's ADA_SQL, then it's inner

------------------------------------------------------------------
--
-- FIND_END_OF_STATEMENT advance pointers to the end of the current statement
--
-- advance pointers to the semicolon at the end of the current statement
-- if we're already at the end just return, if we have to read further into
-- the line read into the current string so on output it will contain
-- a semicolon

------------------------------------------------------------------
--
-- GOT_END_OF_STATEMENT determine if we're at the end of the current statement
--
-- check to see if we're currently pointing at the ; which is
-- the end of the line

------------------------------------------------------------------
```

```
--
-- GET_CONSTANT does the current token match this constant
--
-- if the string in temp string matches the asked for constant and update is
-- true then read the next token and return valid as it was on input,
-- if string doesn't match constant return valid = false

-------------------------------------------------------------------
--
-- GET_CONSTANT_MAYBE if the current token matches this constant advance
-- pointers past this tokan
--
-- if the string in temp string matches the asked for constant and update is
-- true then read the next token and return valid as it was on input
-- and return got as true,
-- if not return valid as entered and got as false

-------------------------------------------------------------------
--
-- ADJUST_USER_SCHEMA manipulate the schema name to the format we want
--
-- adjust the inputed user name to upper case, lower case or leave it as it
-- if the name input by the user has an .ADA or .A, or whatever is the
-- extention for this system as defined in ddliodefs, extention, remove it

-------------------------------------------------------------------
--
-- CHARACTER_STRINGS_MATCH if the two strings match regardless of case
-- return true
```

3.2.67 File DDLSUB2S.ADA

```
-- SUBROUTINES_2_ROUTINES contain some of the subroutines used by the ddl reader

-- SPLIT_IDENT_2_PACKS split up a string containing an identifier and
-- possibly up to two qualifying packages

-- FIND_IDENTIFIER_DESCRIPTOR given an identifier return it's
-- identifier_descriptor

-- FIND_FULL_NAME_COMPONENT_DESCRIPTOR given an identifier's
-- identifier_descriptor and a full package name and a table name return the
-- full_name_descriptor of a component or null if it's not found

-- FIND_FULL_NAME_DESCRIPTOR given an identifier's identifier_descriptor
-- and a full package name return the full_name_descriptor or null if
```

```
-- it's not found

-- GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR given the identifier descriptor
-- and potential package names look for a full name descriptor

-- FIND_FULL_NAME_DESCRIPTOR_VISIBLE given the schema unit and identifier's
-- descriptor find the full name descriptor

-- BASE_TYPE_INTEGER find out if the base type of the identifier is an integer

-- LOCATE_PREVIOUS_IDENTIFIER given an identifier, possibly qualified return
-- it's identifier descriptor and it's full name descriptor

-- STRING_TO_INT convert a character representation of a number to an integer

-- BASE_TYPE_CHAR given a full_name descriptor find out if it's base type
-- is character

-- IS_IDENTIFIER_NULL_OR_UNIQUE is the identifier of a _not_null or
-- _not_null_unique type

-- IN_ADA_SQL_PACKAGE are we currently within a sub package named ADA_SQL

-- ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS add identifier and full name
-- descriptors to the chains for this identifier name

-- ADD_NEW_IDENT_AND_OR_FULL_NAME_COMPONENT_DESCRIPTORS add an identifier and
-- full name descriptor for the component (database column) name
```

3.2.68 File DDLSUB2B.ADA

```
-- SUBROUTINES_2_ROUTINES contain some of the subroutines used by the ddl reader

------------------------------------------------------------------
--
-- SPLIT_IDENT_2_PACKS split up a string containing an identifier and
-- possibly up to two qualifying packages

------------------------------------------------------------------
--
-- FIND_IDENTIFIER_DESCRIPTOR given an identifier return it's
-- identifier_descriptor

------------------------------------------------------------------
--
-- FIND_FULL_NAME_COMPONENT_DESCRIPTOR given an identifier's
```

```
-- identifier_descriptor and a full package name and a table name return the
-- full_name_descriptor of a component or null if it's not found

------------------------------------------------------------------
--
-- FIND_FULL_NAME_DESCRIPTOR given an identifier's identifier_descriptor
-- and a full package name return the full_name_descriptor or null if
-- it's not found

------------------------------------------------------------------
--
-- GET_READY_TO_FIND_FULL_NAME_DESCRIPTOR given the identifier descriptor
-- and potential package names look for a full name descriptor
--
-- given the identifier descriptor and possible known outter and inner
-- packages and possible trying outter and inner packages set up to create
-- the full package name to look for in the full name descriptors.
-- there must be at least one outter and one inner package.  the known ones
-- must be used if available and if there are corresponding try ones they
-- better match.

------------------------------------------------------------------
--
-- FIND_FULL_NAME_DESCRIPTOR_VISIBLE given the schema unit and identifier's
-- descriptor find the full name descriptor
--
-- given current schema, identifier's descriptor and either no package names,
-- both the inner and outter package name or only the inner package name
-- of only the outter if its one of the special (database, standard,
-- cursor_definition) find the full name descriptor that would be
-- visible from current schema.  First choice is current package.  If no match
-- then next choice is from packages currently used (it's already been
-- established at this point that we're two levels deep into packages unless
-- we're doing one of the special ones).  If it isn't found yet then we have
-- to search the withed list, but in that case the full package name better
-- be described.

------------------------------------------------------------------
--
-- BASE_TYPE_INTEGER find out if the base type of the identifier is an integer

------------------------------------------------------------------
--
-- LOCATE_PREVIOUS_IDENTIFIER given an identifier, possibly qualified return
-- it's identifier descriptor and it's full name descriptor
--
-- error 0 = ok
-- error 1 = it is not a valid qualified identifier
-- error 2 = does not split correctly into 2 packages and 1 identifier
```

```
--           maybe invalid nesting of packages
-- error 3 = cannot find identifier by this name
-- error 4 = cannot identify unique full name identifier of this name

-------------------------------------------------------------------
--
-- STRING_TO_INT convert a character representation of a number to an integer

-------------------------------------------------------------------
--
-- BASE_TYPE_CHAR given a full_name descriptor find out if it's base type
-- is character

-------------------------------------------------------------------
--
-- IS_IDENTIFIER_NULL_OR_UNIQUE is the identifier of a _not_null or
-- _not_null_unique type

-------------------------------------------------------------
--
-- IN_ADA_SQL_PACKAGE are we currently within a sub package named ADA_SQL
--
-- we also return true if the current package is one of the standard ones

-------------------------------------------------------------------
--
-- ADD_NEW_IDENT_AND_OR_FULL_NAME_DESCRIPTORS add identifier and full name
-- descriptors to the chains for this identifier name
--
-- identifier descriptor may already exist, but if not create one
-- full name descriptor will not already exist, create it

-------------------------------------------------------------------
--
-- ADD_NEW_IDENT_AND_OR_FULL_NAME_COMPONENT_DESCRIPTORS add an identifier and
-- full name descriptor for the component (database column) name
--
-- identifier descriptor may already exist, but if not create one
-- full name descriptor will not already exist, create it
```

3.2.69 File DDLSUB3S.ADA

```
-- SUBROUTINES_3_ROUTINES contain some of the subroutines used by the ddl reader

-- BREAK_DOWN_SUBTYPE_INDICATOR break a subtype indicator down into small
-- usable parts
```

```
-- SUBTYPE_INDICATOR_IS_ENUMERATION validate the subtype indicator from
-- an enumeration declaration

-- LOCATE_ENUMERATION_LITERAL return the position and descriptor of the
-- given literal if it appears in the given type descriptor

-- SUBTYPE_INDICATOR_IS_INTEGER validate the subtype indicator from
-- an integer declaration

-- SUBTYPE_INDICATOR_IS_FLOAT validate the subtype indicator from
-- a floating point declaration

-- SUBTYPE_INDICATOR_IS_STRING validate the subtype indicator from
-- a string declaration

-- INSERT_SUBTYPE_INDICATOR_INFORMATION into the descriptor
```

3.2.70 File DDLSUB3B.ADA

```
-- SUBROUTINES_3_ROUTINES contain some of the subroutines used by the ddl reader

------------------------------------------------------------------------
--
-- BREAK_DOWN_SUBTYPE_INDICATOR break a subtype indicator down into small
-- usable parts
--
-- on entry temp_string should contain the previous identifier of the
-- subtype indicator.  If that type is:
--   unconstrained array - may or may not specify a range and we will return
--                         got_array_index, array_index_lo and array_index_hi
--   constrained array - must specify nothing else
--   integer - may specify a range, return got_integer_range, integer_ragne_lo
--             and integer_range_hi
--   real - may specify a digits and or a range, return got_float_digits,
--          float_digits, got_float_range, float_range_lo and float_range_hi
--   enumeration - may specify a range, return got_enum_range, enum_range_lo,
--                 and enum_range_hi
--   record - invalid
--
--  errors returned:
--    1   the previous identifier was invalid
--    2   the previous identifier was a component
--    3   the previous identifier was a record
--    4   for enumeration range not found but something bogus there
--    5   for enumeration range literals are incorrect
--    6   for integer range not found but something bogus there
```

```
--     7   for integer range integersare incorrect
--     8   for float expecting digits or range or ; found none
--     9   for float digits integers are incorrect
--    10   for float range integers are incorrect
--    11   for string range not found but something bogus there
--    12   for string range is incorrect
--    13   for string range was given for a constrained array
--    14   no longer used - for string range was not given for an
--         unconstrained array
--

------------------------------------------------------------------------
--
-- SUBTYPE_INDICATOR_IS_ENUMERATION validate the subtype indicator from
-- an enumeration declaration
--
-- on entry temp_string should contain either ; or RANGE
-- if ; then just return valid=true
-- if range then it must be followed by two enumeration literal range
-- specifiers.  They must be located in the parent (type_des) and ordered
-- correctly, if so return them, if not error
--
--  errors returned:
--    4    range not found but something bogus there
--    5    range literals are incorrect
--

-- first we either have ; or RANGE

-- now find first range literal

-- now find .. between literals

-- now find range literal 2

-- now we should be at the end of the statement

-- now find out if the literals belong to the parents

------------------------------------------------------------------------
--
-- LOCATE_ENUMERATION_LITERAL return the position and descriptor of the
-- given literal if it appears in the given type descriptor

------------------------------------------------------------------------
--
-- SUBTYPE_INDICATOR_IS_INTEGER validate the subtype indicator from
-- an integer declaration
--
```

```
-- on entry temp_string should contain either ; or RANGE
-- if ; then just return valid=true
-- if range then it must be followed by two integer range
-- specifiers.  They must fuall within the range of the parent (type_des)
-- and be ordered correctly, if so return them, if not error
--
--  errors returned:
--    6    range not found but something bogus there
--    7    range integers are incorrect
--

-- first we either have ; or RANGE

-- now find lo range

-- now find .. between integers

-- now find hi range

-- now we should be at the end of the statement

-- now find out if the range is valid with the parents

------------------------------------------------------------------------
--
-- SUBTYPE_INDICATOR_IS_FLOAT validate the subtype indicator from
-- a floating point declaration
--
-- on entry temp_string should contain either ; or DIGITS or RANGE
-- if ; then just return valid=true
-- if digits then it must be followed by an integer
-- if range then it must be followed by two floats
-- They must fall within the digits and range of the parent (type_des)
-- and be ordered correctly, if so return them, if not error
--
--  errors returned:
--     8    expecting digits or range or ; found none
--     9    digits is incorrect
--    10    range is incorrect

-- first we either have ; or DIGITS or RANGE

-- process DIGITS here

-- process range here

-- now find lo range

-- now find .. between floats
```

```
-- now find hi range

-- now find out if the range is valid with the parents

-- now we should be at the end of the statement

------------------------------------------------------------------------
--
-- SUBTYPE_INDICATOR_IS_STRING validate the subtype indicator from
-- a string declaration
--
-- on entry temp_string should contain either ; or (
-- if ; then just return valid=true
-- if ( then it must be followed by a range and )
-- Range must fall within the range of the parent (type_des)
-- and be ordered correctly, if so return them, if not error
--
--  errors returned:
--    11    range not found but something bogus there
--    12    range is incorrect
--    13    range was given for a constrained array
--    14    no longer used - range was not given for an unconstrained array

-- first we either have ; or (
-- if constrained parent and range supplied = error
-- if unconstrained parent may or may not have range

-- now find lo range

-- now find .. between integers

-- now find hi range

-- now we should be at the end of the statement find );

-- now find out if the range is valid with the parents

------------------------------------------------------------------------
--
-- INSERT_SUBTYPE_INDICATOR_INFORMATION into the descriptor
--
```

3.2.71 File DDLSUB4S.ADA

```
-- SUBROUTINES_4_ROUTINES contain some of the subroutines used by the ddl reader
```

```
-- WITH_USE_SCHEMA_DEFINITION tell me if we've withed and used the package
-- "schema definitions" and if any other package were withed and/or used,
-- not counting "standard"

-- IS_AUTH_ID_UNIQUE return true if the authorization identifier is uniqye
-- and false if it's not.

-- VALIDATE_NULL_UNIQUE_CONSTRAINTS make sure that a _not_null or
-- _not_null_unique identifier is a subtype of another identifier

-- NULL_UNIQUE_NAMES_THE_SAME if we lop off the suffixes are the identifiers
-- the same

-- SET_UP_WITH_USE_STANDARD_FOR_SCHEMA all schema units are set up to with and
-- use "standard" as a default

-- ADD_NEW_ENUM_LIT add a new enumeration literal to the literal chain and to
-- the full name literal chain

-- FIND_EXISTING_ENUM_LIT given an enumeration literal return it's
-- enumeration literal descriptor

-- ADD_NEW_ENUM_LIT_FOR_DERIVED add to the literal chains for a type derived
-- from an enumeration type
```

3.2.72 File DDLSUB4B.ADA

```
-- SUBROUTINES_4_ROUTINES contain some of the subroutines used by the ddl reader

------------------------------------------------------------------------------
--
-- WITH_USE_SCHEMA_DEFINITION tell me if we've withed and used the package
-- "schema definitions" and if any other package were withed and/or used,
-- not counting "standard"

------------------------------------------------------------------------------
--
-- IS_AUTH_ID_UNIQUE return true if the authorization identifier is uniqye
-- and false if it's not.
--
-- Also print error message if necessary

------------------------------------------------------------------------------
--
-- VALIDATE_NULL_UNIQUE_CONSTRAINTS make sure that a _not_null or
-- _not_null_unique identifier is a subtype of another identifier
```

```
--
-- given a subtype descriptor, whose NOT_NULL and NOT_UNIQUE variables reflect
-- the parents, determine if the subtype is more constrained than the parent.
-- also if constraints are involved then the basic name, without suffixes,
-- must be the same.

-------------------------------------------------------------------------------
--
-- NULL_UNIQUE_NAMES_THE_SAME if we lop off the suffixes are the identifiers
-- the same

-------------------------------------------------------------------------
--
-- SET_UP_WITH_USE_STANDARD_FOR_SCHEMA all schema units are set up to with and
-- use "standard" as a default
--
-- if this schema is "STANDARD" then don't do anything
-- if we haven't already withed "STANDARD" then with it
-- if we haven't already used "STANDARD"

-------------------------------------------------------------------------
--
-- ADD_NEW_ENUM_LIT add a new enumeration literal to the literal chain and to
-- the full name literal chain
--
-- the enumeration literal descriptor may already exist, if not create one
-- the full enumeration literal descriptor will not already exist, create it

-----------------------------------------------------------------
--
-- FIND_EXISTING_ENUM_LIT given an enumeration literal return it's
-- enumeration literal descriptor

-------------------------------------------------------------------------------
--
-- ADD_NEW_ENUM_LIT_FOR_DERIVED add to the literal chains for a type derived
-- from an enumeration type
```

3.2.73 File CHARTOS.ADA

```
-- chartos.ada - CONVERT_CHARACTER_TO_COMPONENT's post process data structures
-- and routines for generating the necessary routines

-- Ada/SQL permits strings to be arrays with components of any type derived
-- from CHARACTER.  When processing data returned from the database, Ada/SQL
-- stores strings as STRINGs.  For passing it back to an application program,
-- this returned data is converted to its program array type by an INTO
```

```
-- procedure instantiated from a generic string INTO procedure.  There is one
-- string INTO procedure instantiated for each program string type that may be
-- returned to the application program.

-- The generic INTO procedure converts the returned database STRING into the
-- program array type character by character, explicitly converting each
-- program component to type CHARACTER.  (This conversion is unnecessary for
-- program array types of CHARACTER, but I figured that the INTO procedure
-- would probably have to be looking at each character of the result anyway,
-- in order to decode where a particular column result stops and the next one
-- starts, so why not let it call the conversion routine in all instances?  If
-- the conversion routine is INLINEd, then it doesn't generate any code
-- anyway.  I did not bother with pragma INLINE in the example, but it could
-- be easily added since the entire generated package is now [will soon be]
-- magically produced by computer.)

-- This explicit conversion is performed by a function called CONVERT_-
-- CHARACTER_TO_COMPONENT, which is a generic formal subprogram to the generic
-- INTO procedure.  The application scanner generates the required functions
-- named CONVERT_CHARACTER_TO_COMPONENT, so that each INTO procedure
-- instantiation uses the correct component conversion function by default (no
-- actual parameter need by supplied to the instantiation for the CONVERT_-
-- CHARACTER_TO_COMPONENT generic formal subprogram.)

-- There is one CONVERT_CHARACTER_TO_COMPONENT function generated for each
-- type, including CHARACTER, used as the component type of a string program
-- type that is retrieved from the database.  Since the functions rely on the
-- fact that the component type is derived from CHARACTER, they cannot be
-- merely instantiated from generics, but must be completely written.  In
-- what follows, type_name represents the fully qualified name of a component
-- type.  If the type is defined in a DDL package, type_name will be of the
-- form library_unit.ADA_SQL.type_simple_name.  If the type is defined in a
-- predefined package, type_name will be of the form library_unit.type_-
-- simple_name.  This includes STANDARD.CHARACTER -- the hand-generated
-- package for the runtime example used a type_name of CHARACTER, but
-- STANDARD.CHARACTER is easier to program (no need to check for special
-- case), and may be used.

-- The specification of each CONVERT_CHARACTER_TO_COMPONENT function is:
--
--   function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
--    return type_name;

-- The corresponding body is:
--
--   function CONVERT_CHARACTER_TO_COMPONENT ( C : CHARACTER )
--    return type_name is
--   begin
--     return type_name ( C );
```

```
--   end CONVERT_CHARACTER_TO_COMPONENT;

-- Where type_name was CHARACTER, the hand-generated package for the runtime
-- example did not apply the conversion function in the body, saying just
-- "return C;".  There is certainly no harm in applying a type conversion
-- function to STANDARD.CHARACTER, and this may be done, rather than program
-- for the special case.

-- The only information required to produce each CONVERT_CHARACTER_TO_-
-- COMPONENT function is the fully qualified name of the type involved.  This
-- information is found in the ACCESS_FULL_NAME_DESCRIPTOR for the type, and
-- it is a pointer to that data structure that is passed to CONVERT_-
-- CHARACTER_TO_COMPONENT.REQUIRED_FOR to indicate that a component conversion
-- function is to be generated for the indicated type.  CONVERT_CHARACTER_TO_-
-- COMPONENT.REQUIRED_FOR is called whenever it is determined that a component
-- conversion function is required; it automatically avoids generating
-- duplicate functions.

-- There are two post processing steps for the CONVERT_CHARACTER_TO_COMPONENT
-- functions: producing the specifications and producing the bodies.  These
-- two steps are performed by CONVERT_CHARACTER_TO_COMPONENT.SPEC_POST_-
-- PROCESSING and CONVERT_CHARACTER_TO_COMPONENT.BODY_POST_PROCESSING.
```

3.2.74 File CHARTOB.ADA

```
-- chartob.ada - CONVERT_CHARACTER_TO_COMPONENT's post process data structures
-- and routines for generating the necessary routines

-- data structures to form a chain of array component types that need routines
-- generated to convert characters to the component types

-- comparison of DDL_DEFINITIONS.ASSESS_FULL_NAME_DESCRIPTOR on left and right

-- avoid generating duplicate functions

   -- Order list by fully-qualified component type name.

-- produce the specification for the convert character to component routines

-- produce the body for the convert character to component routines
```

3.2.75 File COLUMNS.ADA

```
-- COLUMN_LIST data structures and for making a chain of database columns

-- data structure for making a chain of the database columns

-- add a new column to the chain of database columns
```

3.2.76 File COLUMNB.ADA

```
-- COLUMN_LIST data structures and for making a chain of database columns

-- add a new column to the chain of database columns
```

3.2.77 File COMPTOS.ADA

```
-- comptos.ada - CONVERT_COMPONENT_TO_CHARACTER's post process data structures
-- and routines for generating the necessary routines

-- Ada/SQL permits strings to be arrays with components of any type derived
-- from CHARACTER.  In its internal data structures, Ada/SQL stores strings as
-- STRINGs.  An array program value is converted to its internal
-- representation by a function instantiated from a generic string conversion
-- function.  There is one string conversion function instantiated for each
-- program string type that must be converted to internal representation.

-- If the component type of the program string type is not CHARACTER, then the
-- string conversion function for that type must convert the program value
-- character by character, explicitly converting each program component to
-- type CHARACTER.  This explicit conversion is performed by a function called
-- CONVERT_COMPONENT_TO_CHARACTER, which is a generic formal subprogram to
-- the generic string conversion function.  The application scanner generates
-- the required subprograms named CONVERT_COMPONENT_TO_CHARACTER, so that each
-- string conversion function instantiation uses the correct component
-- conversion function by default (no actual parameter need be supplied to
-- the instantiation for the CONVERT_COMPONENT_TO_CHARACTER generic formal
-- subprogram.)

-- There is one CONVERT_COMPONENT_TO_CHARACTER function generated for each
-- type, other than CHARACTER, used as the component type of a string program
-- type that must be converted to internal representation.  Since the
-- functions rely on the fact that the component type is derived from
```

```
-- CHARACTER, they cannot be merely instantiated from generics, but must be
-- completely written.  In what follows, type_name represents the fully
-- qualified name of a component type.  If the type is defined in a DDL
-- package, type_name will be of the form library_unit.ADA_SQL.type_simple_-
-- name.  If the type is defined in a predefined package, type_name will be
-- of the form library_unit.type_simple_name.

-- The specification of each CONVERT_COMPONENT_TO_CHARACTER function is:
--
--   function CONVERT_COMPONENT_TO_CHARACTER ( C: type_name )
--    return CHARACTER;

-- The corresponding body is:
--
--   function CONVERT_COMPONENT_TO_CHARACTER ( C: type_name )
--    return CHARACTER is
--   begin
--     return CHARACTER ( C );
--   end CONVERT_COMPONENT_TO_CHARACTER;

-- The only information required to produce each CONVERT_COMPONENT_TO_-
-- CHARACTER function is the fully qualified name of the type involved.  This
-- information is found in the ACCESS_FULL_NAME_DESCRIPTOR for the type, and
-- it is a pointer to that data structure that is passed to CONVERT_-
-- COMPONENT_TO_CHARACTER.REQUIRED_FOR to indicate that a component conversion
-- function is to be generated for the indicated type.  CONVERT_COMPONENT_TO_-
-- CHARACTER.REQUIRED_FOR is called whenever it is determined that a component
-- conversion function is required; it automatically avoids generating
-- duplicate functions.

-- There are two post processing steps for the CONVERT_COMPONENT_TO_CHARACTER
-- functions: producing the specifications and producing the bodies.  These
-- two steps are performed by CONVERT_COMPONENT_TO_CHARACTER.SPEC_POST_-
-- PROCESSING and CONVERT_COMPONENT_TO_CHARACTER.BODY_POST_PROCESSING.
```

3.2.78 File COMPTOB.ADA

```
-- comptob.ada - CONVERT_COMPONENT_TO_CHARACTER's post process data structures
-- and routines for generating the necessary routines

-- data structures to form a chain of array component types that need routines
-- generated to convert component types to characters

-- comparison of DDL_DEFINITIONS.ASSESS_FULL_NAME_DESCRIPTOR on left and right

-- avoid generating duplicate functions
```

```
   -- Order list by fully-qualified component type name.

-- produce the specification for the convert component to character routines
```

Distribution List for IDA Memorandum Report M-461

NAME AND ADDRESS	NUMBER OF COPIES
Sponsor	
Mr. James Robinette WIS JPMO/DXP Room 5B19 Washington, D.C. 20330-6600	2 copies
Other	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
Mr. Fred Friedman P.O. Box 576 Annandale, VA 22003	1 copy
Mr. Kevin Heatwole 5124 Harford Lane Burke, VA 22015	1 copy
Ms. Kerry Hilliard 7321 Franklin Road Annandale, VA 22003	1 copy
CSED Review Panel	
Dr. Dan Alpert, Director Center for Advanced Study University of Illinois 912 W. Illinois Street Urbana, Illinois 61801	1 copy
Dr. Barry W. Boehm TRW Defense Systems Group MS 2-2304 One Space Park Redondo Beach, CA 90278	1 copy
Dr. Ruth Davis The Pymatuning Group, Inc. 2000 N. 15th Street, Suite 707 Arlington, VA 22201	1 copy

NAME AND ADDRESS	NUMBER OF COPIES
Dr. Larry E. Druffel Software Engineering Institute Carnegie-Mellon University Pittsburgh, PA 15231-3890	1 copy
Dr. C.E. Hutchinson, Dean Thayer School of Engineering Dartmouth College Hanover, NH 03755	1 copy
Mr. A.J. Jordano Manager, Systems & Software Engineering Headquarters Federal Systems Division 6600 Rockledge Dr. Bethesda, MD 20817	1 copy
Mr. Robert K. Lehto Mainstay 302 Mill St. Occoquan, VA 22125	1 copy
Mr. Oliver Selfridge 45 Percy Road Lexington, MA 02173	1 copy
IDA	
General W.Y. Smith, HQ	1 copy
Mr. Philip Major, HQ	1 copy
Dr. Robert E. Roberts, HQ	1 copy
Dr. Jack Kramer, CSED	1 copy
Dr. Robert I. Winner, CSED	1 copy
Dr. John Salasin, CSED	1 copy
Mr. Bill R. Brykczynski, CSED	1 copy
Ms. Katydean Price, CSED	2 copies
IDA Control & Distribution Vault	3 copies